ПРИЛОЖЕНИЕ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
ИМЕНИ В.Ф. УТКИНА»

Кафедра «Автоматизация информационных и технологических процессов»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по дисциплине

Б2.В.01.01(У) «Ознакомительная практика»

Направление подготовки
15.03.04 «Автоматизация технологических процессов и производств»

Направленность (профиль) подготовки
«Автоматизация технологических процессов и производств»

Уровень подготовки – академический бакалавриат
Квалификация выпускника – бакалавр
Форма обучения – очная, заочная

Рязань

УДК 004.4'236

Разработка приложений доступа к базам данных в среде Borland Delphi: методические указания к лабораторным работам. Часть 1 / Рязан. гос. радиотехн. ун-т.; сост.: А. В. Алпатов, В. В. Гудзев, О. В. Мельник. Рязань, 2009. 60 с.

Рассматриваются лабораторные работы по курсам «Компьютерные технологии в медицине» и «Современные информационные технологии в микроэлектронике»

Предназначены студентам, обучающимся по специальностям 200401, 210104.

Табл. 12. Ил. 13. Библиогр.: 4 назв.

Визуальное программирование, Delphi, базы данных, объектно-ориентированное программирование, SQL

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра биомедицинской и полупроводниковой электроники Рязанского государственного радиотехнического университета (зав. кафедрой, проф. С.П. Вихров)

Р А Б О Т А № 1

ЗНАКОМСТВО СО СРЕДОЙ BORLAND DELPHI И СОЗДАНИЕ ПРОСТЕЙШИХ ПРИЛОЖЕНИЙ

Цель работы: получить навыки программирования в среде Borland Delphi, познакомиться с основными визуальными компонентами и методами работы с ними.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Введение

Программный пакет Borland Delphi является интегрированной средой разработки (Integrated Development Environment – IDE). Этот термин подразумевает, что пакет представляет собой не просто одну программу, а совокупность программ, библиотек, программных модулей, сервисных утилит, которые позволяют повысить эффективность разработки приложения за счет максимальной автоматизации процесса его проектирования и уменьшения рутинного труда.

Важной особенностью такой среды является возможность создавать программы в стиле визуального конструирования формы в режиме реального времени, путем создания окон приложения и размещения на них компонентов программы.

Работая в режиме визуального конструирования, программист использует специальное окно, которое в терминах среды Delphi называется формой. Форма является прототипом окна создаваемой программы. На форме можно размещать компоненты, которые будут реализовывать нужную функциональность программы. Например, визуальные компоненты (которые отображаются во время выполнения программы): кнопки, строки редактирования, меню или не визуальные, например посредники для работы с базами данных. После размещения компонента на форме Delphi автоматически добавляет необходимые ссылки в файл исходного текста, который связан с данной формой, предоставляя программисту возможность заниматься исключительно функционалом того или иного компонента, не тратя время на его описание. По этой причине среда программирования Delphi относится к классу средств ускоренной разработки программ (общепринятое сокращение RAD –Rapid Application Development).

Запуск среды осуществляется через главное меню рабочего стола – кнопку “Пуск”: Пуск -> Borland Delphi 7-> Delphi 7.

Основные элементы среды Borland Delphi

После загрузки системы на экране появляются шесть основных окон (актуально для Borland Delphi 7).

1. Главное окно с основными интерфейсными элементами, именем проекта по умолчанию Project1.
2. Окно проектирования формы приложения, по умолчанию называется Form1.
3. Окно инспектора объектов Object Inspector, в котором содержатся Свойства (Property) и События (Events) выбранного объекта по умолчанию Form1.
4. Окно редактора кода – по умолчанию Unit1.pas.
5. Окно дерева объектов Object Inspector, в котором иерархически отображаются все объекты принадлежащие Form1.
6. Окно браузера кода, которое связано с окном редактора кода.

Вид и расположение окон можно изменять.

В главном окне реализуются все основные функции управления проектом создаваемой программы. Главное управляющее окно системы Delphi по умолчанию располагается в верхней части экрана. Все элементы главного окна располагаются на специальных панелях:

- Заголовок ;
- Главное меню;
- Панель инструментов;
- Палитра компонентов.

Строка заголовка главного окна отображает имя открытого в данный момент проекта. Строка меню содержит команды, необходимые для разработки и тестирования приложений, и используется так же, как любое стандартное меню MS Windows. Панели инструментов предназначены для выполнения наиболее часто используемых команд, реализуемых главным меню. Палитра компонентов содержит множество вкладок. Каждая из них содержит свой набор компонентов. Общее число компонентов достигает нескольких сотен. Совокупность наборов составляет библиотеку визуальных компонентов (Visual Components Library – VCL). Имеется несколько категорий компонентов, каждая из которых расположена на своей вкладке. С помощью палитры компонентов создаются экземпляры компонентов (объекты) на форме. Первой кнопкой на каждой вкладке палитры компонентов всегда является указатель в виде курсора. Это не компонент, эта кнопка позволяет быстро отказаться от уже выбранного компонента.

Окно инспектора объектов (Object Inspector) отображает свойства какого-либо компонента, активизированного щелчком мыши, или самой формы. Имя активизированного компонента находится под заголовком окна. С помощью инспектора объектов настраивают объекты, используе-

мые в программах. Это же окно используется для выбора и настройки событий, на которые будут реагировать объекты нашей программы. С его помощью создаются или выбираются нужные процедуры обработки.

Свойства и реакции на события. Каждый объект в Delphi, включая окно разрабатываемого приложения, имеет определенные свойства. Например, цвет, размер, отображаемый текст и т.п. Эти свойства можно изменять еще до запуска проектируемой программы на выполнение. В зависимости от изменяемого свойства результат можно просматривать уже сразу.

Реакции на события - это результаты произошедшего системного события, например щелчка мыши, нажатия на кнопку, открытия окна и т.п. Реакцию на событие назначают программно.

Окно проектирования формы приложения определяет внешний вид и интерфейс взаимодействия пользователя с программой, а также содержит на себе все необходимые компоненты, обеспечивающие функциональность программы. В окне проектирования формы происходит сборка программы из компонентов, содержащихся в палитре компонентов.

Главным элементом окна проектирования является форма. Каждое Windows-приложение выполняется в собственном окне. Минимальное количество таких окон равно 1. Среда Delphi при запуске автоматически предлагает пользователю новый проект, открывая пустое (незаполненное) окно под названием Form1, и назначает его главным окном. Главное окно (форма) в проекте может быть только одно. Все другие создаваемые окна будут дочерними. Заготовка первого окна по умолчанию называется Form1. Если в программе будет два окна, то заготовка второго будет называться Form2 и так далее. Возможна замена стандартного названия более подходящим для программы.

Окно редактора кода содержит код программы на языке Object Pascal. Часть исходного программного кода система Delphi формирует автоматически и скрывает от пользователя. На момент первого запуска окно редактора кода называется Unit1. В редакторе кода могут быть открыты сразу несколько файлов. Каждый открытый файл размещается на отдельной закладке-странице. Как правило, в проектах основную работу программист производит именно в этом окне.

Важным понятием среды Delphi является **проект**. Термин “проект” имеет множество толкований. Универсальное определение гласит, что проект – это уникальная, целенаправленная деятельность, имеющая начало и конец во времени, направленная на достижение определённого результата, создание определённого, уникального продукта или услуги. Термин проект происходит от латинского слова *projectus*, что в переводе означает «брошенный вперед».

В контексте разработки программного обеспечения проектом является деятельность программиста или группы программистов, направленная на создание функционально-законченного программного обеспечения. По аналогии с этим определением проект в Delphi – это объединение файлов исходных текстов и настроек среды проектирования, необходимых для решения задачи программирования. Проект включает в себя множество файлов. Наиболее важными являются три файла: файл формы, файл кода и файл проекта.

DPR - файл проекта. Содержит основной код программы, ссылки на все окна (формы) проекта и относящиеся к ним модули. В нем также содержится код инициализации. Имеет одноименное название с проектом.

PAS - pascal файл. Он содержит текст, который вы видите в окне редактора кода так называемого модуля программы, – ваш исходный текст.

DFM - delphi form. Представляет собой файл с полными данными о проектировщике формы. Позиция, размер, расположенные компоненты и пр. Форма приложения является неотъемлемой частью модуля PAS и имеет то же название, что и ваша форма .

DCU - двоичный файл модуля. Имеет одноименное название с модулем.

RES - ресурсный файл. Содержит в себе иконки, значки указателя мыши, картинки, звуки.

DOF, DSK - содержат настройки проекта.

CFG - содержит настройки конфигурации проекта.

EXE - откомпилированная программа. Сохраняется автоматически при запуске проекта на выполнение. Обновляется в момент компиляции. Имеет одноименное название проекта. Полностью самостоятельное приложение.

В процессе работы в среде Delphi могут автоматически создаваться файлы, имеющие расширение, начинающееся символом ~. Это резервные копии файлов, которые создаются при их повторном сохранении.

В процессе компиляции программы файлы преобразуются в исполняемый exe-файл, который по умолчанию создается в той же папке, в которой расположен файл проекта. В проекте могут быть задействованы несколько форм, а также дополнительные модули и файлы ресурсов. Для сохранения всего проекта нужно воспользоваться пунктом главного меню Save All.

Компоненты среды Delphi

Главной ценностью среды Delphi являются компоненты. Библиотека визуальных компонентов предоставляет программисту огромное разнообразие созданных разработчиками Delphi программных заготовок, которые немедленно или после несложной настройки готовы к работе в рамках вашей программы. Компоненты характеризуются важным свойством: они включают в себя программный код и все необходимые для его применения

данные, что избавляет программиста от рутинной работы по «изобретению велосипедов» — нет нужды писать то, что уже написано другими, причем проверено и отлажено. Как правило, в программном пакете среды Delphi поставляется множество компонентов, рассчитанных на самые разные аспекты применения — от простых компонентов, создающих поясняющие надписи, до сложных текстовых процессоров или инструментов принятия решений. Если по каким-либо причинам в Delphi нет компонента с нужной функциональностью, его можно создать средствами самой среды Delphi и включить затем в библиотеку VCL. Существует и другой, весьма распространенный среди программистов вариант — использование ресурсов сети Интернет, где предлагаются бесплатные, условно-бесплатные и платные компоненты, созданные специально для Delphi как профессиональными программистами, так и любителями.

Несмотря на различие форм и функций компонентов, их можно разбить на группы и классифицировать. Как уже ранее говорилось, все компоненты можно разделить на визуальные и невизуальные. Визуальные, в свою очередь, делятся на оконные элементы управления и графические элементы управления.

Визуальные компоненты являются элементами управления, которые добавляются на форму: это кнопки, поля ввода, картинки, панели и т.д. Те из них, что предназначены для ввода данных, могут принимать фокус ввода, обеспечивая управление с клавиатуры.

Невизуальные компоненты служат в основном для обеспечения более удобного доступа к определенным функциям системы. При добавлении их на форму на ней появляется иконка с изображением компонента, невидимая во время исполнения программы. В этом и заключается их «невизуальность». Несмотря на это, некоторые из них служат для обеспечения визуальных по природе эффектов. Например, к таким относятся компоненты меню и диалоги.

Визуальные компоненты составляют интерфейс программы. Визуальные компоненты имеют много общих свойств и событий, связанных, прежде всего, с их визуальным отображением на форме.

Часть свойств отвечает за положение на форме: отступ слева, отступ сверху, высота, ширина, выравнивание.

Некоторые определяют внешний вид: цвет, шрифт, заголовок. Некоторые задают поведение компонента во время исполнения программы: доступен, виден, курсор, подсказка.

Механизм событий позволяет связать действия пользователя с необходимыми действиями программы. События определяют всю структуру программы, поскольку фактически весь код, то есть собственно работа программиста, пишется в обработчиках событий.

В системе Windows действия пользователя обычно сводятся к нажатиям мыши в определенных местах экрана, нажатию клавиш клавиатуры, переключению между окнами и компонентами. В соответствии с этим су-

ществуют и события, реагирующие на нажатие мыши и клавиатуры, переключение фокуса, активацию окон.

Кроме того, для каждого компонента могут существовать характерные для него события, на которые могут потребоваться реакция и обработка, например событие изменения текста в поле ввода.

Использование инспектора объектов (F11)

Любой размещаемый на форме компонент характеризуется некоторым набором параметров: положением, размером, цветом и т. д. Часть этих параметров, например, положение и размеры компонента, программист может изменять, манипулируя компонентом в окне формы. Каждый элемент управления, как и сама форма, обязательно имеет название. Название по умолчанию формирует сама система. Кроме того, практически все элементы управления также характеризуются текстовой информационной строкой - надписью, которая первоначально совпадает с названием элемента. Например, кнопка на форме получает название и надпись Button1. Эти надписи можно изменять.

Для изменения этих и других параметров предназначено окно инспектора объектов. Это окно содержит две вкладки — Properties (Свойства) и Events (События). Вкладка Properties служит для установки нужных свойств компонента, вкладка Events позволяет определить реакцию компонента на то или иное событие. Совокупность свойств отображает видимую сторону компонента: положение относительно левого верхнего угла рабочей области формы, его размеры и цвет, шрифт и текст надписи на нем и т. п.; совокупность событий — его поведенческую сторону: будет ли компонент реагировать на щелчок мыши или на нажатие клавиш, как он будет вести себя в момент появления на экране или в момент изменения размеров окна и т. п.

Каждая вкладка окна инспектора объектов представляет собой таблицу из двух колонок: левая колонка содержит название свойства или события, а правая — конкретное значение свойства или имя подпрограммы, обрабатывающей соответствующее событие.

Строки таблицы выбираются щелчком мыши, и в них могут отображаться простые или сложные свойства. К простым относятся свойства, определяемые единственным значением — числом, строкой символов, значением True (Истина) или False (Ложь) и т. п. Например, свойство Caption (Заголовок) представляется строкой символов, свойства Height (Высота) и Width (Ширина) — числами, свойство Enabled (Доступность) — значениями True или False. Сложные свойства определяются совокупностью значений. Слева от имени таких свойств указывается значок «+», а щелчок мышью на нем приводит к раскрытию списка составляющих сложного свойства. Чтобы закрыть раскрытый список, нужно щелкнуть на значке «-», в который после щелчка превращается значок «+».

Щелчок мышью на правой колонке любой строки таблицы активизирует указанное в ней значение свойства, при этом в правом конце строки может появиться кнопка со стрелкой или кнопка с тремя точками. Щелчок на кнопке с тремя точками приводит к появлению на экране диалогового окна, с помощью которого устанавливается значение сложного свойства. Щелчок на кнопке со стрелкой ведет к раскрытию выпадающего списка возможных значений простого свойства.

В верхней части окна инспектора объектов располагается раскрывающийся список всех помещенных на форму компонентов. Поскольку форма сама по себе является компонентом, ее имя также присутствует в этом списке.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В данной лабораторной работе вы должны получить навыки создания простейших приложений в среде Delphi. Разобраться с функциями и интерфейсом основных окон среды. Научить управлять свойствами и событиями компонентов и использовать их для расширения функциональности программы.

Лабораторная работа проходит в три этапа:

Этап 1. Самостоятельная подготовка к лабораторной работе. Получение допуска преподавателя к выполнению лабораторной работы.

Этап 2. Выполнение лабораторной работы.

Этап 1. Подготовка к лабораторной работе

1. По таблице 1 выберите компоненты для изучения. Используя справку среды Delphi, литературные и Интернет-источники, в том числе и поисковые серверы (google.ru, yandex.ru, rambler.ru и т.п.), выполните описание свойств (Properties) выбранных компонентов с указанием их назначения.

Пример фрагмента описания компонента TForm.

…

Свойство Caption	Содержимое свойства определяет текст заголовка Формы.
Свойство Name	Содержимое свойства задает имя объекта для программного обращения к нему. По умолчанию – Form1

…

2. Подготовьте бланк отчета о проведении лабораторной работы. В пункте Этап1 приведите описание свойств выбранных компонентов. Отчет должен готовиться каждым студентом индивидуально.

Таблица 1. Визуальные компоненты из библиотеки VLC Borland Delphi

Номер группы	Название компонента	Закладка окна визуальных компонент
1	TButton TBevel	Standard Additional
2	TPanel TTimer	Standard Systems
3	TPopupMenu TTrackBar	Standard Win32
4	TLabel TStatusBar	Standard Win32
5	TEdit TSpeedButton	Standard Additional
6	TMemo TPaintBox	Standard Systems
7	TCheckBox TImage	Standard Additional
8	TListBox TProgressBar	Standard Win32
9	TRadioButton TScrollBox	Standard Additional
10	TListBox TProgressBar	Standard Win32
11	TComboBox TRadioGroup	Standard Standard
12	TMainMenu	Standard

Этап 2. Выполнение лабораторной работы

2.1. Знакомство с интерфейсом

Проведем исследование возможностей выбранных визуальных компонент. Для этого создадим проект с общими для всех групп функциями. С помощью созданного проекта вы получите возможность изменения свойств ваших компонент и создадите демонстрационный пример в виде запускаемого файла (exe-файл).

Известно, что каждое Windows-приложение выполняется в собственном окне. Минимальное количество таких окон равно 1. Среда Delphi при запуске автоматически предложит вам новый проект (Project), открыв пустое (незаполненное) окно под названием Form1 и назначив его главным окном (главное окно – это окно, в котором происходит основная работа с программой, как правило, оно появляется самым первым после загрузки программы, не считая заставки).

Итак, после запуска среды Delphi вы увидите уже открытый новый проект для создания приложения. Этот проект можно сразу запустить на выполнение. Это можно сделать несколькими путями: нажатием на кнопку F9, выбрав соответствующий пункт "Run" в меню "Run" главного окна среды Delphi, или нажатием соответствующей кнопки на панели инструментов главного окна (кнопка зеленого цвета в стиле “Play”). Эта операция называется “запуск на компиляцию программы и выполнение”. После этого происходит сравнительно недолгий процесс компиляции (перевод исходного файла в расширение “EXE”. Далее этот файл, в случае успешного создания, запускается на выполнение. Во время выполнения из множества окон Delphi остаются только главное окно и окно редактора кода.

Скомпилированная программа имеет вид стандартного Windows-окна с заголовком Form1, со стандартными кнопками: свернуть, на весь экран, закрыть. Его можно позиционировать, изменять размер. По умолчанию окно отображается в том же виде, что и во время разработки, имеет те же размеры и позицию на экране.

Закроем запущенную программу – Delphi автоматически переведет окна в то состояние, которое было до запуска проекта на выполнение. Снова на экране появляются инспектор объектов и редактор формы.

Теперь нужно сохранить проект на диске. Настоятельно рекомендуется сохранять на диске новый проект сразу, поскольку он находится в памяти компьютера.

Для этого создайте на диске, который вам укажет преподаватель, папку с номером вашей группы, в ней создайте папку с номером работы (например, Lab1).

Теперь сохраните все файлы проекта. Для этого выберите подпункт Save All, находящийся в пункте меню File главного окна среды Delphi. Вам предложат сохранить модуль программы Unit1 (исходный текст, связанный с формой Form1). Вы можете переименовать его, например, в main.pas или выбрать имя по смыслу основной задачи данного модуля. Замечание: сколько существует окон в вашей программе, столько и будет модулей, поэтому и рекомендуется сохранять каждый проект в отдельный каталог. Далее вам будет предложено сохранить проект как Project1: т.е. задать название всего проекта. Как будет называться проект, под таким же именем и будут создаваться исполняемые EXE файлы. Название проекта также можно изменять.

Обратите внимание! Названия файлов должны состоять из одного слова или слов, написанных английскими буквами, цифры допустимы только начиная со второго символа, пробелы недопустимы (используйте в таких случаях знак подчеркивания).

Теперь, когда вы научились запускать и сохранять проект, приступим к созданию вашей первой программы.

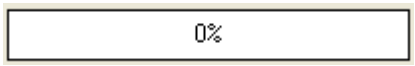
2.2. Создание демонстрационной программы

Программа будет состоять из наиболее часто используемых компонентов и функций, с помощью которых вы будете изменять свойства ваших компонентов.

Описание действий при разработке программы для компактности приводится в виде двух таблиц (данная форма представления будет актуальна и для всех остальных работ данного цикла). В таблице 2 указывается, какие компоненты вы должны разместить на форме и в каком порядке. Для каждого компонента указываются название и закладка, где его нужно искать. В таблице 3 указывается, какие свойства компонентов нужно изменить, какой исходный код необходимо написать для того или иного события.

Рассмотрим демонстрационный пример изучения свойств компонента Gauge (стилизация под измерительный прибор) на вкладке Samples.

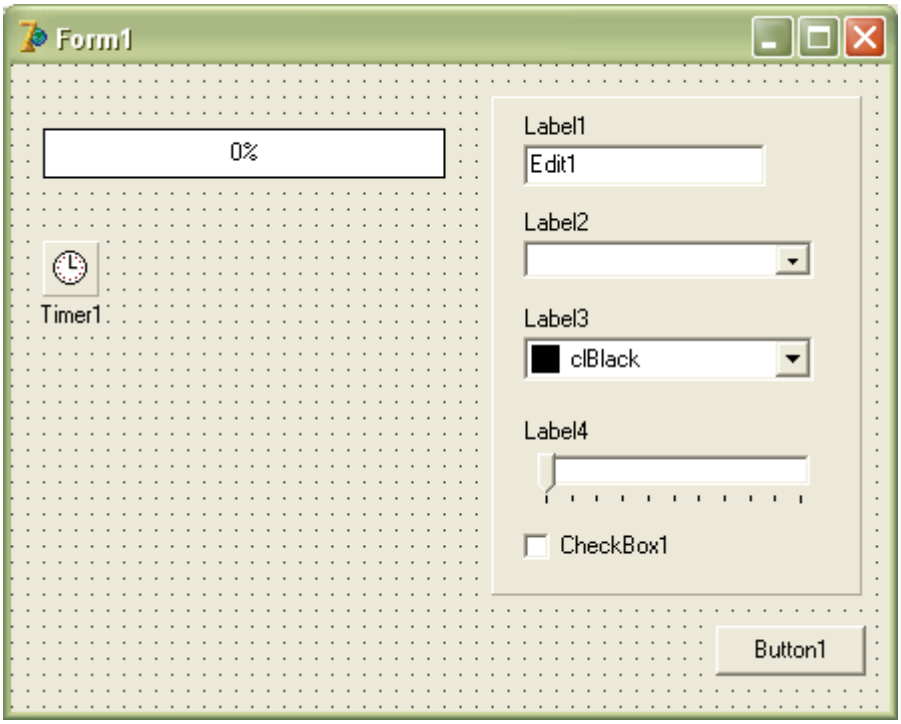
Таблица 2. Перечень компонентов

Название компонента	**Описание**	**Закладка**	**Внешний вид**
Gauge	Индикатор величины	Samples	
Panel1	Панель (элемент дизайна)	Samples	
Label1	Текстовая метка	Standard	
Edit1	Строка редактирования	Standard	
Label2	Текстовая метка	Standard	
Box1	Раскрывающийся список	Standard	
Label3	Текстовая метка	Standard	
ColorBox1	Раскрывающийся список цветов	Additional	
CheckBox1	Элемент выбора “флажок”	Standard	
Label4	Текстовая метка	Standard	
TrackBar1	“Ползунок”	Win32	
Button1	Стандартная кнопка	Standard	
Timer1	Таймер	System	

В итоге форма должна принять вид, показанный на рисунке. Все компоненты кроме Gauge, Timer и Button должны лежать на компоненте Panel, для этого размер Panel нужно увеличить.

После того как все компоненты будут размещены на форме, необходимо повторно сохранить проект. Запустите проект на компиляцию и выполнение. Если все выполнено правильно, произойдет запуск вашей программы.

Пока функциональность компонентов не описана, они ничего не “умеют“ делать: выпадающие списки пустые, кнопка не реагирует на нажатие. Закройте программу с помощью стандартных кнопок. Выполните действия согласно таблице 3. При программировании реакции на событие переход в окно исходного текста осуществляется двойным щелчком на выбранном событии.



Форма демонстрационного примера

Таблица 3

Выделенный объект (на форме)	Вкладка окна Object Inspector (F11)	Имя свойства/ имя события	Действие
Form1	Properties	Caption	Установка имени формы «Демо проект»
	Events	OnCreate	При создании формы установить начальное значение свойств некоторых компонентов Combobox1.ItemIndex := 0; CheckBox1.Checked := False; Gauge1.Progress := 0; Edit1.Text := 100

Продолжение табл. 3

Gauge1	Properties	MinValue	Установка минимального значения: 0 (по умолчанию)
Label1	Properties	Caption	Установка текстовой метки «Максимальное значение»
Edit1	Properties	Text	Удалить все содержимое
Label2	Properties	Caption	Установка текстовой метки «Режим отображения»
Combobox1	Properties	Items	Заполнить раскрывающийся список. Нажать … . В появившемся окне с верхней строки написать: *Горизонтальный* *Полукруглый* *Круглый* *Текстовый* *Вертикальный* Нажать “ОК”
	Properties	Style	Установить режим отображения списка - фиксированный csOwnerDrawFixed
	Events	OnChange	Событие при изменении текущего пункта списка. Исходный код: Case Combobox1.ItemIndex of 0: Gauge1.Kind := gkHorizontalBar; 1: Gauge1.Kind := gkNeedle; 2: Gauge1.Kind := gkPie; 3: Gauge1.Kind := gkText; 4: Gauge1.Kind := gkVerticalBar; end;
Label3	Properties	Caption	Установка текстовой метки «Цвет индикатора»
ColorBox1	Events	OnChange	Событие при изменении текущего пункта списка цветов. Исходный код: Gauge1.ForeColor := ColorBox1. Selected;
Label4	Properties	Caption	Установка текстовой метки «Размер по вертикали»
TrackBar1	Properties	Max	Установка максимального значения: 200
	Properties	Min	Установка минимального значения: 30
	Events	OnChange	Событие при изменении положе-

Окончание табл. 3

			ния ползунка. Исходный код: Gauge1.Height := TrackBar1.Position;
CheckBox1	Properties	Caption	Установка комментария флажка «Запуск индикации»
Button1	Properties	Caption	Установка имени кнопки «Выход»
	Events	OnClick	Событие при нажатии на кнопку. Исходный код: Close;
Timer1	Properties	Interval	Установка шага срабатывания таймера в миллисекундах: 250
	Events	OnTimer	Событие при срабатывании таймера. Исходный код: If Not CheckBox1.Checked then exit; Gauge1.MaxValue := StrToInt(Edit1.Text); Gauge1.Progress := Gauge1.Progress + 1; If Gauge1.Progress > Gauge1.MaxValue then Gauge1.Progress := 0;

После того как все изменения будут выполнены, вам необходимо сохранить проект.

Запустите проект на компиляцию и выполнение (F9). Если все выполнено правильно и нет ошибок, произойдет запуск вашей программы. Проверьте работу каждого компонента.

1. Установив флажок “Запуск индикации”, вы должны увидеть как по циклу изменяется положение индикатора от 0 до 100 %. При снятии галочки изменение индикатора останавливается.
2. Выбирая различные цвета с помощью раскрывающегося списка цветов, вы должны увидеть, как изменяется цвет индикатора.
3. Изменяя режимы отображения с помощью раскрывающегося списка, вы должны увидеть, как изменяется стиль индикатора.
4. Изменяя цифру максимального значения, вы должны увидеть как изменяется скорость заполнения индикатора до 100 % при условии, что флажок “Запуск индикации” установлен.

Покажите ваш результат преподавателю.

1.3. Задание на самостоятельную работу

По аналогии с приведенным демонстрационным проектом напишите (измените исходную) версию программы для демонстрации возможностей одного из двух компонентов, которые вы описали на этапе 1.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие вы знаете основные элементы среды Delphi? Перечислите их.
2. Что такое инспектор объектов? Какие у него функции?
3. Из чего состоит программный проект, создаваемый в Delphi? Каково назначение каждого файла проекта?
4. Какие вы знаете визуальные компоненты в закладке Standard? Перечислите их функции.
5. Какие вы знаете визуальные компоненты в закладке Additional? Перечислите их функции.
6. Какие вы знаете визуальные компоненты в закладке Win32? Перечислите их функции.
7. Какие вы знаете события компонентов и как происходит реакция компонентов на эти события?
8. Каковы назначение и отличие секций interface и implementation исходного кода (файл *.pas)?
9. Как повлияет на выполнение программы удаление строки Edit1.Text := 100?
10. Как повлияет на выполнение программы удаление условия If Not CheckBox1.Checked then exit?
11. Какой цвет будет у индикатора, если строку Gauge1.ForeColor := ColorBox1.Selected изменить на Gauge1.Color := ColorBox1.Selected?
12. Что означает запись StrToInt(Edit1.Text)?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Поисковая система Google.ru. Запросы: “Delphi”, “Описание компонентов delphi 7”, “электронные учебники delphi 7”
2. Delphi. Программирование на языке высокого уровня: учебник для вузов / В.В. Фараонов. СПб.: Питер, 2003. 640 с.
3. Введение в DELPHI, Epsylon Technologies
   http://www.citforum.ru/programming/32less/index.shtml

Р А Б О Т А № 2

ЗНАКОМСТВО С ОСНОВНЫМИ ПРИНЦИПАМИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В СРЕДЕ BORLAND DELPHI

Цель работы: получить навыки программирования на объектно-ориентированном языке Object Pascal в среде Borland Delphi путем создания пользовательских визуальных компонентов.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Введение в объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — это парадигма программирования (стиль программирования, иначе говоря — некоторый цельный набор идей и рекомендаций, определяющих стиль написания программ), в которой предметная область представляется системой структур данных, каждая из которых представляет некий отдельный предмет (объект), относящийся к своему типу с его внутренними свойствами и действиями над ним.

Основными концепциями ООП являются понятия объектов и классов.

Класс (Class) — это тип, описывающий устройство объектов. В общем случае понятие «класс» подразумевает некоторое поведение и способ представления. Понятие «объект» подразумевает нечто, что обладает определённым поведением и способом представления. Говорят, что объект — это экземпляр класса. Таким образом, класс можно сравнить с шаблоном, по которому создается множество объектов. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области. Например, если вам необходимо включить в состав вашей программы кнопку, которая бы реагировала на нажатие мыши, вы можете создать класс “Кнопка” и в виде абстрактного описания определить ее цвет, надпись, размеры, положение, способ рисования на экране, внешний вид и т.д., а также описать, как реагировать на нажатие той или иной кнопки мыши, клавиши клавиатуры. Определив класс, вы можете в программе создавать произвольное число объектов – кнопок, используя в качестве шаблона описание в виде класса. Таким образом, под классом подразумевается некая сущность, которая задает некоторое общее поведение для всех объектов.

Класс определяет для объекта правила, с помощью которых с объектом могут работать другие объекты (обычно это делается с помощью определения методов класса). Классы могут находиться друг с другом в различных отношениях, таких как Наследование или Агрегация.

Фактически объектно-ориентированное программирование чаще всего сводится к созданию некоторого количества классов, описанию связей между этими классами и их свойств и дальнейшей реализации полученных классов.

Пример описания класса в среде Delphi выглядит так:

```
type
        TMyObject = class
        MyField: Integer;
        function MyMethod: Integer;
end;
```

где TMyObject – название класса (в Delphi все названия классов всегда называются с буквы T), MyField – поле класса, MyMethod – метод класса.

Объект (Object) – это фрагмент адресного пространства вычислительной системы, имеющий определенную структуру и появляющийся при создании *экземпляра класса*. Объект состоит из данных и связанных с ними процедур (которые называются методами). Объект может наследовать характеристики порождающего объекта. Это означает, что структура данных нового объекта включает структуру данных порождающего объекта, а также новые данные. Кроме того, новый объект может вызывать все процедуры порождающего объекта, а также те процедуры методов, которые в нем описываются.

Объект создается в результате вызова специального метода, который инициализирует объект, – **конструктора**. Созданный экземпляр уничтожается другим методом – **деструктором**. До передачи управления телу конструктора происходит собственно создание объекта — под него отводится память, значения всех полей обнуляются. Далее выполняется код конструктора, написанный программистом для инициализации экземпляров данного класса. Таким образом, хотя на первый взгляд синтаксис конструктора схож с вызовом процедуры (не определено возвращаемое значение), на самом деле конструктор – это функция, возвращающая созданный и инициализированный объект.

Необходимо отметить, что в среде Delphi объект — это динамическая структура. Переменная-объект содержит не данные, а ссылку на данные объекта – адрес памяти. Помимо выделения памяти, конструктор, как правило, решает задачу присваивания полям объекта начальных значений, т. е. осуществляет инициализацию объекта.

Пример создания объекта на базе класса TMyObject

```
AMyObject := TMyObject.Create;
```

После того как объект выполнит все возложенные на него функции, его необходимо удалить, т.е. очистить занимаемую им память. Для этого используется вызов деструктора AMyObject.Destroy.

Поле (Field) – переменная, связанная с классом или объектом. Все данные объекта хранятся в его полях и предназначены для хранения данных во время работы экземпляра класса (объекта).. Доступ к полям осу-

ществляется по их имени. Обычно тип данных каждого поля задаётся в описании класса, членом которого является поле. Ограничений на тип полей в классе не предусматривается. В описании класса поля должны предшествовать методам и свойствам. Обычно поля используются для обеспечения выполнения операций внутри класса. Все поля предназначены для использования внутри класса. Однако класс должен каким-либо образом взаимодействовать с другими классами или программными элементами приложения. В подавляющем большинстве случаев класс должен выполнить с некоторыми данными определенные действия и представить результат. Для получения и передачи данных в классе применяются свойства..

Свойство (Property) — это атрибут, который составляет внешнюю индивидуальность объекта и помогает описать его некоторую характеристику. С точки зрения программы понятия поля и свойства схожи, в общем случае это переменные, в которых хранится некоторая информация. Основное отличие заключается в том, что поля скрыты от пользователя объекта, поскольку определяют его функциональность, а свойства, наоборот, видны пользователю и позволяют ему изменять поведение объекта. Фактически свойства – абстрактный элемент, который скрывает от пользователя вызов метода, что упрощает логику работы с объектом. Например, обычная кнопка в окне приложения обладает такими свойствами, как цвет, размеры, положение. Для экземпляра класса "кнопка" значения этих атрибутов задаются с помощью свойств – специальных переменных, определяемых ключевым словом property. Цвет может задаваться свойством Color, размеры — свойствами Width и Height и т. д. Все изменения этих свойств можно осуществлять только с помощью вызовов соответствующих методов.

Метод (Method) — это объявленная в классе функция или процедура, которая используется для работы с полями и свойствами класса. От обычных процедур и функций методы отличаются тем, что им при вызове передается указатель на тот объект, который их вызвал. Поэтому обрабатываться будут данные именно того объекта, который вызвал метод. Согласно принципам инкапсуляции обращаться к свойствам класса можно только через его методы. Пользователь объекта не должен иметь прямой доступ к его свойствам, поскольку их некорректное определение может привести к нарушению работы объекта, поэтому любое изменение свойства сопровождается вызовом соответствующих методов. Например, доступ к значению свойства кнопки, определяющей ее цвет – Color, осуществляется через вызовы методов GetColor и SetColor. Таким образом, программист, который работает с объектом, меняет какое-либо свойство, а объект автоматически вызывает нужный метод, который изменяет поле внутри объекта или вызывает какую-либо процедуру. При этом компилятор самостоятельно переведет обращение к свойству Color к вызовам методов GetColor или SetColor в зависимости от типа операции – чтения или

записи соответственно. Таким образом внешне свойство выглядит в точности как обычное поле, но за всяким обращением к нему могут стоять нужные вам действия. Например, если у вас есть объект, представляющий собой квадрат на экране, и его свойству "цвет" вы присваиваете значение "белый", то произойдет немедленная перерисовка, приводящая реальный цвет на экране в соответствие со значением свойства. Выполнение этой операции осуществляется методом, который связан с установкой значения свойства "цвет". В методах, входящих в состав свойств, могут осуществляться проверка устанавливаемой величины на попадание в допустимый диапазон значений и вызов других процедур, зависящих от вносимых изменений в поведение объекта.

Событие (Event) — это метод, описывающий реакцию объекта на внешнее воздействие. В операционной системе реального времени, такой как Microsoft Windows, события – это один из главных принципов работы приложений. События возникают тогда, когда один программный компонент передает другому сообщение. В сообщении содержатся информация о типе события и данные, которые нужно передать.

Сообщения бывают от устройств, например мыши и клавиатуры, и от программ и объектов, из которых они состоят. В процессе работы программы возникают множество событий, связанных с функционированием операционной системы: перерисовка, изменение размеров окна, открытие и закрытие окна и т.п. Например, если пользователь решил закрыть окно (визуальный объект) программы, а редактируемый текст остался не сохраненным, то программист должен в событии этого объекта “Закрытие окна”, которое обязательно возникнет после нажатия крестика в правом верхнем углу, написать программный код, который задает вопрос о необходимости сохранения и действует далее по указанию пользователя (сохраняет в файл или выходит из программы без изменения).

В результате при использовании объектно-ориентированного подхода программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов).

Парадигма ООП имеет характерные механизмы реализации, перечислим их.

1. Абстракция данных — подход к обработке данных по принципу чёрного ящика. Данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня. Объекты представляют собой неполную информацию о реальных сущностях предметной области. Получаемые в этом случае модели адекватны решаемой задаче, работать с ними намного удобнее, чем с низкоуровневым описанием всех возможных свойств и реакций объекта.

2. Наследование — возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка (иногда его называют суперклассом) и добавлением, при необходимости, новых свойств и мето-

дов. Наследование призвано отобразить такое свойство реального мира, как иерархичность.

3. Полиморфизм — явление, при котором классы-потомки могут изменять реализацию метода класса-предка, сохраняя его сигнатуру (таким образом, сохраняя неизменным интерфейс класса-предка). Это позволяет обрабатывать объекты классов-потомков как однотипные объекты, несмотря на то, что реализация методов у них может различаться.

4. Инкапсуляция — это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик – пользователь класса должен видеть и использовать только интерфейс (т. е. список декларируемых свойств и методов) класса и не вникать в его внутреннюю реализацию. Таким образом, инкапсуляция позволяет объединить данные и код в объект и скрыть реализацию объекта от пользователя.

Рассмотрим эти механизмы более подробно.

Инкапсуляция. Как правило, объект — это сложная конструкция, свойства и поведение составных частей которой находятся во взаимодействии. При создании программных объектов подобные ситуации можно моделировать, связывая со свойствами необходимые методы. Понятие инкапсуляции соответствует этому механизму.

Классическое правило объектно-ориентированного программирования утверждает, что для обеспечения надежности нежелателен прямой доступ к полям объекта: чтение и обновление их содержимого должны производиться посредством вызова соответствующих методов. Это правило и называется инкапсуляцией.

Инкапсуляция — свойство языка программирования, позволяющее скрывать реализацию объекта от пользователя. При этом пользователю предоставляется только спецификация (интерфейс) объекта. Пользователь может взаимодействовать с объектом только через интерфейс, т.е. между объектом и пользователем существует посредник, который очень хорошо знает объект и не позволяет пользователю выполнить недопустимые операции с изменением свойств объекта, просто лишь поменяв значения переменной, ведь неизвестно, как это повлияет на работу объекта.

Наследование. Этот простой принцип означает, что если вы хотите создать новый класс, лишь немного отличающийся от старого, то совершенно нет необходимости в переписывании заново уже существующих полей и методов. Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class).Унаследованные от класса-предка поля и методы доступны в дочернем классе; если имеет место совпадение имен методов, то говорят, что они перекрываются.

При наследовании все параметры и методы, описанные в родительском классе, переходят без изменений в класс-потомок. Но не всегда тре-

буется такое буквальное копирование. Иногда просто необходимо переопределить уже существующий метод или параметр. Для того чтобы при вызове одинакового метода различные потомки вели себя по-разному, требуется замещение метода, описанного в классе-предке.

В среде Delphi родительским является класс TObject, который инкапсулирует основное поведение всех классов и отвечает за выделение и освобождение памяти при создании и удалении объектов.

Полиморфизм. Полиморфизм в объектно-ориентированном программировании – это способность объекта выбирать правильный метод в зависимости от типа данных, полученных в сообщении. Сущность полиморфизма лучше всего рассматривать на конкретном примере.

Пусть у нас имеется некий класс, который должен хранить данные. Назовем его TField (поле текста). Опишем его так:

```
type
        TField = class
        function GetData: string; virtual; abstract;
end;
```

Этот класс содержит виртуальный метод GetData, который на данный момент не определен.

Создадим два его потомка — TStringFieid для строк, TIntegerField для целых чисел. В виде исходного текста это будет выглядеть так.

1. Создадим потомка класса TField для хранения текста методом наследования

```
type
        TStringFieid = class(TField)
        FData : string; // новое свойство - тестовое поле
        function GetData: string; override;
end;
```

где GetData – переопределенный (override) метод по выводу текстовых данных

```
function TStringFieid.GetData;
 begin  Result := FData;   end;
```

2. Создадим потомка класса TField для хранения целых чисел

```
type
        TIntegerField = class(TField)
        FData : Integer; // поле с целыми числами
        function GetData: string; override;
end;
```

где GetData - переопределенный (override) метод по выводу целых чисел путем их предварительного преобразования (функция IntToStr()) в текстовый тип данных.

```
function TIntegerField.GetData;
begin Result := IntToStr(FData); // преобразовать число в строку end;
```

Теперь создадим процедуру ShowData, которая будет присваивать заголовку (Caption) главного окна (MainForm) программы значение свойства, которое задал пользователь через FData

```
procedure ShowData(MyField : TField);
begin  MainForm.Caption := MyField.GetData; end;
```

В этом примере классы содержат разнотипные поля данных FData и могут только сообщить о значении этих данных текстовой строкой (с помощью метода GetData). Внешняя по отношению к ним процедура ShowData получает объект в виде параметра и показывает эту строку.

В процедуре ShowData параметр описан как TFieid – это значит, что в нее можно передавать объекты классов и TStringField, и TIntegerField и любого другого потомка класса TFieid.

Например,

ShowData (TStringField) – вывод текста,

ShowData (TIntegerField) – вывод целого числа.

Но «чей» метод GetData при этом будет вызван? Тот, который соответствует классу фактически переданного объекта. Этот принцип и называется полиморфизмом. Заметим, что независимо от типа аргумента (в первом случае это символьная строка, во втором – целое число, в третьем – вещественное число) обработка происходит единообразно.

Таким образом объектно-ориентированный принцип разработки дает много преимуществ. Например, каждый объект объединяет свою структуру данных с процедурой, используемой для работы с экземплярами структуры данных Это позволяет устранить в коде программы внутренние зависимости, которые могут быстро привести к тому, что этот код будет трудно обслуживать. Объекты могут также наследовать из порождающего объекта структуры данных и другие характеристики, что позволяет сэкономить усилия и обеспечить прозрачное использование для многих целей больших фрагментов кода.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В данной лабораторной работе вы должны получить навыки использования возможностей объектно-ориентированного программирования при создании собственных визуальных компонентов. В процессе работы вы должны усвоить принципы инкапсулации, наследования и полиморфизма, разобраться с понятием объектов и классов.

Лабораторная работа проходит в два этапа.

Этап 1. Самостоятельная подготовка к лабораторной работе. Получение допуска преподавателя к выполнению лабораторной работы.

Этап 2. Выполнение лабораторной работы.

Этап 1. Подготовка к лабораторной работе

В качестве базового класса, с которым вы будете работать и на основе которого вы создадите собственный класс, выберем знакомый по лабораторной работе 1 индикатор величины Gauge.

В лабораторной работе 1 был использован экземпляр класса TGauge – компонент Gauge1 (см. в Unit1.pas строку Gauge1: TGauge, которая появилась автоматически после добавления компонента Gauge на форму Form1).

Теперь вам предстоит создать новый класс TMyGauge, который должен унаследовать все свойства, методы и события своего родителя TGauge и получить новые свойства, которых у него не было.

На этапе подготовки к лабораторной работе вам необходимо ознакомиться с “устройством” компонента TGauge. Исходный код программы компонента находится по следующему пути: C:\Program Files\Borland\Delphi7\Source\Samples\gauges.pas (путь по умолчанию, при инсталляции дистрибутива Delphi с настройками по умолчанию). Необходимо отметить, что в каталоге Source хранятся все исходные коды всех компонент Delphi, что является хорошим примером для программистов, начинающих создавать свои компоненты. Если по каким-либо причинам этого файла не оказалось, вы можете воспользоваться поиском в Интернете по ключевому слову “Gauges.pas”.

В отчете вам необходимо привести полный текст исходного кода компонента, причем в секции interface выделить прямоугольными рамками фрагменты, описывающие поля, свойства, методы, и процедуры, обеспечивающие функциональность компонента. Данный отчет будет являться допуском к лабораторной работе.

Этап 2. Выполнение лабораторной работы

Возможности объектно-ориентированного программирования позволяют довольно гибко изменять функциональность базового класса, оставаясь в рамках его предназначения. Так, переопределив (override) метод Paint – отрисовка, можно полностью поменять внешний вид компонента, например стилизовать под любой стрелочный прибор, индикатор заряда батареи или уровня приема радиосигнала. В данном случае мы ограничимся только небольшими изменениями внешнего вида, например добавим возможность изменения цвета и толщины рамки по периметру компонента. По умолчанию она отрисовывается толщиной только 1 пиксель и всегда черного цвета.

Для этого добавим новые свойства и изменим процедуру рисования компонента (таблица 1).

Таблица 1

Свойство	**Назначение**
FrameColor, тип TColor;	Определяет цвет прямоугольной рамки
WidthtFrame	Определяет толщину рамки в пикселях

Окончание табл. 1

Метод	Исходный код

Paint	inherited Paint;
	with Canvas do begin
	Canvas.Brush.Style := bsClear;
	Canvas.Pen.Color := PenColor;
	Canvas.Pen.Width := PenWidth;
	if BorderStyle = bsSingle then
	Canvas.Rectangle(0,0,Width, Height);
	end;

Рассмотрим подробнее исходный код метода Paint. Служебное слово inherited указывает, что данный метод полностью наследует код аналогичного метода родительского компонента, т.е. внешний вид остается неизменным – мы только добавляем команду рисования рамки поверх предварительно отрисованного компонента. В результате они просто накладываются.

По правилам объектно-ориентированного программирования для реализации функций рисования существует специальный класс – TCanvas. Русскоязычные программисты на Delphi называют его “канвой”.

Этот класс – основа графической подсистемы Delphi. Он объединяет в себе холст – область рисования, рабочие инструменты (шрифт – Font, перо – Pen и кисть – Brush), а также набор функций по рисованию типовых геометрических фигур. Холст рисования можно представить в виде координатной сетки с шагом, равным одному пикселю (графической точке). Все геометрические фигуры рисуются текущим пером. Те из них, которые можно закрашивать, закрашиваются с помощью текущей кисти. Кисть и перо при этом имеют текущий цвет.

Канва не является компонентом, но она присутствует в качестве свойства во многих других компонентах, которые должны уметь нарисовать себя и отобразить какую-либо информацию. Класс TCanvas есть и у нашего объекта, с его помощью мы производим настройку инструментов рисования:

- Canvas.Brush.Style := bsClear — бесцветная кисть (без заполнения);
- Canvas.Pen.Color := PenColor — цвет пера в зависимости от значения поля PenColor;
- Canvas.Pen.Width:= PenWidth — толщина пера в зависимости от значения поля PenWidth,

И далее, если свойство BorderStyle установлено пользователем как bsSingle рисуем прямоугольник (Rectangle) во весь размер компонента (0,0,Width, Height) от верхнего левого угла до правого нижнего.

Теперь, разобравшись с особенностями исходного кода, приступим к созданию компонента средствами Delphi.

Для автоматизации этого процесса в среде Delphi предусмотрен специальный мастер.

Выполните следующие действия.

Действие	**Комментарии**
Главное меню Delphi. File -> New -> Other	Создание нового компонента через главное меню. В результате откроется окно с типовыми шаблонами и мастерами
Закладка New -> Иконка “Component” -> Ok	Произойдет запуск мастера создания компонента
Новое окно “New Component”	В поле Ancestor type необходимо задать базовый компонент. Для разрабатываемого компонента базовым типом для компонента является тип TGauge (введите в поле первые буквы названия и вам автоматически будет предлагаться название компонента из существующего списка). **Ancestor type -> TGauge** В поле Class Name нужно ввести имя класса разрабатываемого компонента – в нашем случае TMyGauge. **Class Name - > TMyGauge** В поле Palette Page надо ввести имя палитры компонентов (по умолчанию Samples), в которую после создания компонента будет добавлен его значок. Название палитры можно выбрать из раскрывающегося списка. Если в поле Palette Page ввести имя еще не существующей палитры, то непосредственно перед добавлением компонента палитра с указанным именем будет создана (появится новая закладка). **Palette Page -> Samples** В поле Unit file name находится автоматически сформированное имя файла модуля создаваемого компонента. Delphi присваивает модулю компонента имя, совпадающее с именем типа компонента, но без буквы T. Измените путь в папку Lab2 (в папке с номером вашей группы)
Нажмите кнопку “OK”	В результате в файле с именем mygauge.pas сохранится заготовка модуля компонента. Сохранный файл отобразится в окне исходного кода. В объявлении нового класса указан только тип родительского класса. В раздел реализации помещена процедура Register, которая используется для регистрации нового класса во время установки созданного программистом компонента в палитру компонентов Delphi

Теперь в сформированное Delphi объявление класса нового компонента нужно внести дополнения: объявить поле данных свойства, функцию доступа к полю данных и процедуру установки значения поля данных и свойств. Для этого заполним исходным кодом пустые секции.

Выполните следующие действия согласно Таблице 2.

Таблица 2

Секция/ подсекция		**Комментарии**
Interface	Uses	К списку библиотек добавьте Graphics, StdCtrls, Forms;. Проследите, чтобы названия следовали через запятые и заканчивались точкой с запятой

	Private	PenColor: TColor;
		PenWidth: Integer;
		procedure SetFrameColor(Value: TColor);
		procedure SetPenWidth(Value: integer);
		Объявляем поля - переменные PenColor, PenWidth, в которых будет храниться текущее состояние рамки.
		Объявляем методы, с помощью которых будет происходить изменение полей (согласно принципу инкапсуляции)

	Protected	procedure Paint; override; *Объявляем метод Paint и указываем, что мы хотим изменить (override) его*
	Published	property FrameColor: TColor read PenColor write SetFrameColor; property FrameWidth: integer read PenWidth write SetPenWidth; *Объявляем (published) новые свойства компонента. При этом указываем, как их нужно читать (read) и записывать (write)*

Implementation	*(После процедуры Register)*
	procedure TMyGauge.Paint;
	begin
	inherited Paint;
	with Canvas do begin
	Brush.Style := bsClear;
	Pen.Color := PenColor;
	Pen.Width := PenWidth;
	if BorderStyle = bsSingle then Rectangle(0,0,Width, Height);
	end;
	end;
	procedure TMyGauge.SetFrameColor(Value: TColor);
	begin	

Окончание табл. 2

	if Value <> PenColor then
	begin
	PenColor := Value;
	Refresh;
	end;
	end;
	procedure TMyGauge.SetPenWidth(Value: integer);
	begin
	if Value <> PenWidth then
	begin
	PenWidth := Value;
	Refresh;
	end;
	end;

File -> Save или Ctrl-S.	Сохраните все изменения

Созданный вами новый компонент необходимо установить в систему. Это делается через главное меню Component -> Install Component…

В появившемся окне в поле Unit file name с помощью кнопки Browse… укажите путь до вашего компонента в папке Lab2. Нажмите кнопку “ОК”, и ваш компонент добавится в пользовательскую библиотеку компонентов (package) Borland User Components, специально созданную для нестандартных компонентов пользователя. Данная информация отобразится в появившемся окне Package. Проверьте, чтобы ваш файл находился в разделе Contains (состав библиотеки). Нажмите кнопку “Compile” для компиляции всех файлов библиотеки. Если ваш исходный текст написан правильно, то в появившемся окне компиляции Compile выведется сообщение “Done. Compiled.”, если есть ошибки, то сообщение “Done. There are errors.”. Используя окно редактора текста, устраните все ошибки, сверившись с таблицей 2 (например, забыли запятую, неправильно указали название процедуры, поместили код не в нужную секцию).

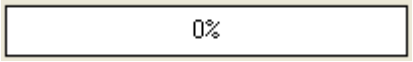
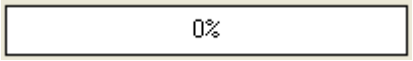
После успешного завершения компиляции проинсталлируйте компонент, нажав кнопку “Install” в окне Package. В результате появится окно с сообщением о регистрации компонента – нажмите “ОК”.

Закройте проект командой File -> Close All. На вопрос о сохранении библиотеки ответьте “Yes”.

Теперь проверим, как работает ваш компонент. Создайте новый проект – File->New -> Application. Разместите на нем компоненты согласно таблице 3.

По умолчанию все компоненты выглядят одинаково, однако в окне Object Inspector у компонента tmygauge1 появились новые свойства: FrameColor и FrameWidth . Измените их и вы увидите, как меняется рамка нового компонента. Сохраните проект в папке Lab2 командой Save All.

Таблица 3

Название компонента	**Описание**	**Закладка**	**Внешний вид**
Gauge	Индикатор величины	Samples	
MyGauge (самый последний)	Индикатор величины	Samples	

Покажите результат преподавателю. В отчете приведите исходный текст базового компонента, нового компонента и “снимок” формы проекта, на котором должно быть видно, что у нового компонента изменилась толщина рамки.

Домашнее задание: Добавьте к вашему компоненту возможность обрисовывать еще две формы – облако и дождь

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое класс?
2. Что такое объект?
3. Что такое инкапсуляция, наследование? Прокомментируйте на примере исходного текста нового компонента.
4. Что такое полиморфизм? Приведите пример.
5. Дайте объяснения понятиям поля, метода, свойства. Чем свойство отличается от поля? Прокомментируйте на примере исходного текста нового компонента.
6. Для чего предназначается класс TCanvas? Перечислите его основные методы (воспользуйтесь справкой Delphi - F1).
7. Объясните назначение секций исходного текста: private, protected, public, published.
8. Что произойдет, если в процедуре Paint убрать строку inherited Paint?
9. Что произойдет, если строку if BorderStyle = bsSingle then Rectangle(0,0,Width, Height) заменить на Rectangle(0,0,Width, Height)?
10. Что произойдет, если в строку Brush.Style := bsClear заменить на Brush.Style := bsSolid ?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Поисковая система Google.ru, запросы: “объектно-ориентированное программирование”, “ Delphi и ООП”, “реализация графики в Delphi”
2. Избачков Ю.С, Петров В.Н. Информационные системы: учебник для вузов. 2-е изд. СПб.: Питер. 2006. 656 с.

3. Delphi. Программирование на языке высокого уровня: учебник для вузов / В.В. Фараонов. СПб.: Питер, 2003. 640 с.

Р А Б О Т А № 3

СОЗДАНИЕ ТАБЛИЦ БАЗЫ ДАННЫХ В СРЕДЕ BORLAND DELPHI

Цель работы: изучить основные подходы и термины, принятые при моделировании данных в виде таблиц, получить навыки создания таблиц базы данных в среде Borland Delphi.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Моделирование данных

Любая прикладная информационная система (например, складской учет, продажа билетов и т.п.) требует создания в памяти компьютера динамически обновляемой модели внешнего мира с использованием единого хранилища - базы данных. Полезное содержимое базы данных определяется предметной областью. Предметная область - часть реального мира, подлежащая изучению с целью организации управления и, в конечном счете, автоматизации. Предметная область представляется множеством фрагментов, например предприятие – цехами, дирекцией, бухгалтерией и т.д. Каждый фрагмент предметной области характеризуется множеством объектов и процессов, использующих объекты, а также множеством пользователей, характеризуемых различными взглядами на предметную область.

Словосочетание "динамически обновляемая" означает, что соответствие базы данных текущему состоянию предметной области обеспечивается не периодически, а в режиме реального времени.

Отличительной чертой баз данных следует считать то, что данные хранятся совместно с их описанием, при этом то, каким образом база данных описывает заданную предметную область, определяется заранее выбранной моделью данных.

Любая модель данных должна содержать три компонента.

1. Структура данных – описание точки зрения пользователя на представление данных о предметной области.
2. Набор допустимых операций, выполняемых на структуре данных. Модель данных предполагает как минимум наличие языка определения данных, описывающего структуру их хранения, и языка манипулиро-

вания данными, включающего операции извлечения и модификации данных.

3. Ограничения целостности - механизм поддержания соответствия данных предметной области на основе формально описанных правил, допустимых для данной модели.

В процессе исторического развития теории баз данных использовались следующие модели данных: иерархическая, сетевая, реляционная, объектно-ориентированная (см. лабораторную работу N 2). Кратко рассмотрим эти модели.

Иерархическая модель данных. Организация данных иерархического типа определяется в терминах: элемент, агрегат, запись (группа), групповое отношение, база данных:

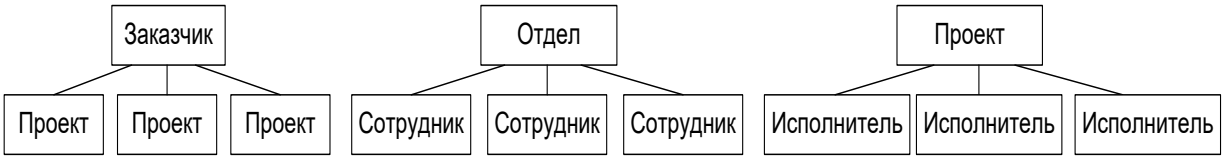
- *атрибут* (элемент данных) - наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем;
- *запись* – именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. Экземпляр записи - конкретная запись с конкретным значением элементов;
- *групповое отношение* – иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) - подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути.

При графическом изображении групповые отношения изображают дугами ориентированного графа, а типы записей – вершинами.

Для групповых отношений в иерархической модели обеспечиваются автоматический режим включения и фиксированное членство. Это означает, что для запоминания любой некорневой записи в БД должна существовать ее родительская запись (подробнее о режимах включения и исключения записей сказано в разделе о сетевой модели). При удалении родительской записи автоматически удаляются все подчиненные.

В иерархических моделях должна автоматически поддерживаться целостность ссылок между предками и потомками, при этом всегда выполняется основное правило: никакой потомок не может существовать без своего родителя.



а б в

Рис. 1. Иерархическое моделирование предметной области “Работа над проектом”: а – описание поступающих проектов, б – описание структуры отдела, в - описание работы над проектом в виде группового отношения

Иерархическая модель проста для понимания, но имеет ряд недостатков, ограничивающих ее возможности. Рассмотрим пример, приведенный на рис. 1. Предметная область “Работа над проектом” имеет иерархическую природу: есть заказчик, который предлагает проекты, есть сотрудники, которые над ними работают, есть разные отделы, в которых работают сотрудники. Из этого примера четко видны недостатки иерархических БД:

- частично дублируется информация между записями “Сотрудник” и “Исполнитель” (такие записи называют парными), потому что интуитивно понятно, что исполнитель – это сотрудник, который участвует в проекте;
- чтобы узнать, какие исполнители работают над данным проектом, необходимо проверить все(!) групповые отношения “Исполнитель”–“Проект” для поиска нужного проекта;
- иерархическая модель реализует отношение между исходной (родительской) и дочерней записью по схеме один-ко-многим, то есть одной родительской записи может соответствовать любое число дочерних. Допустим, что исполнитель может принимать участие более чем в одном проекте (т.е. возникает связь типа многие-ко-многим). В этом случае в базу данных необходимо ввести еще одно групповое отношение, в котором запись “Исполнитель” будет являться исходной записью, а “Проект” – дочерней (перевернуть схему “в” с ног на голову), при этом возникнет эффект дублирования – запись “Исполнитель” появится одновременно в двух групповых отношениях.

Сетевая модель данных. Организация данных сетевого типа определяется в тех же терминах, что и иерархическая. Она состоит из множе-

ства записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид один-ко-многим. Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом более чем одного группового отношения, т.е один потомок может иметь любое число предков. Пример такого представления данных показан на рис. 2. Потомок “Проект” связан с двумя предками “Заказчик” и “Сотрудник”. Таким образом, можно реализовывать связи “один проект - много сотрудников” и “один сотрудник – много проектов”. Как видно, преодоление недостатков иерархической модели потребовало введения специальных связывающих элементов “Сотрудник в проекте”.

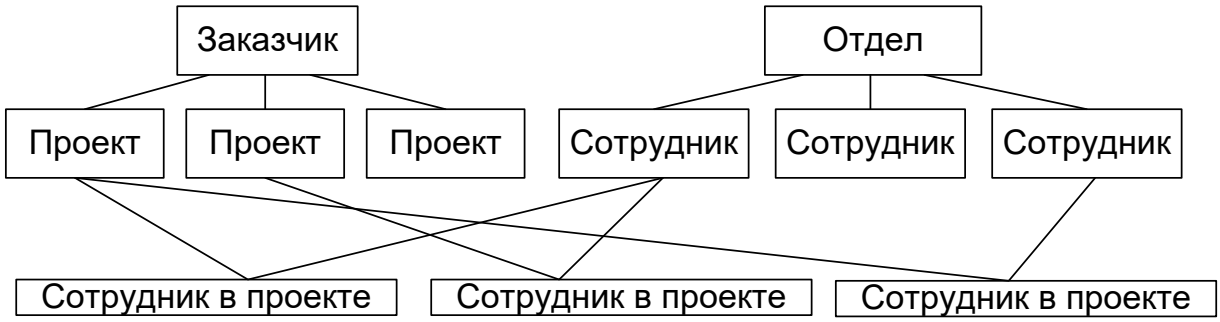


Рис. 2. Сетевое моделирование предметной области
“Работа над проектом”

Достоинством сетевой модели является отсутствие поддержания ограничений целостности, так как все связи жестко задаются в процессе проектирования базы данных, нужно отслеживать только их целостность.

В итоге при использовании сетевой модели получались достаточно сложные структуры, состоящие из "наборов" – поименованных двухуровневых деревьев. "Наборы" соединяются с помощью "записей-связок", образуя цепочки и т.д. Особенно сложность такого представления проявлялась при большом количестве записей и групповых отношений, причем связи фиксировались на физическом уровне и не могли быть изменены в дальнейшем.

Дальнейшие изыскания в области моделирования данных привели к созданию реляционных моделей.

Реляционная модель данных. Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение. Для того чтобы дальнейшее изложение материала было понятным, на рис. 3 в графическом виде представлены все основные понятия. На рисунке видно, что обычным представлением отношения является таблица, заголовком которой является схема отношения, а строками – кортежи отношения-экземпляра; в этом случае имена атрибутов имену-

ют столбцы этой таблицы. Поэтому иногда говорят "столбец таблицы", имея в виду "атрибут отношения". Теперь подробно рассмотрим каждое понятие.

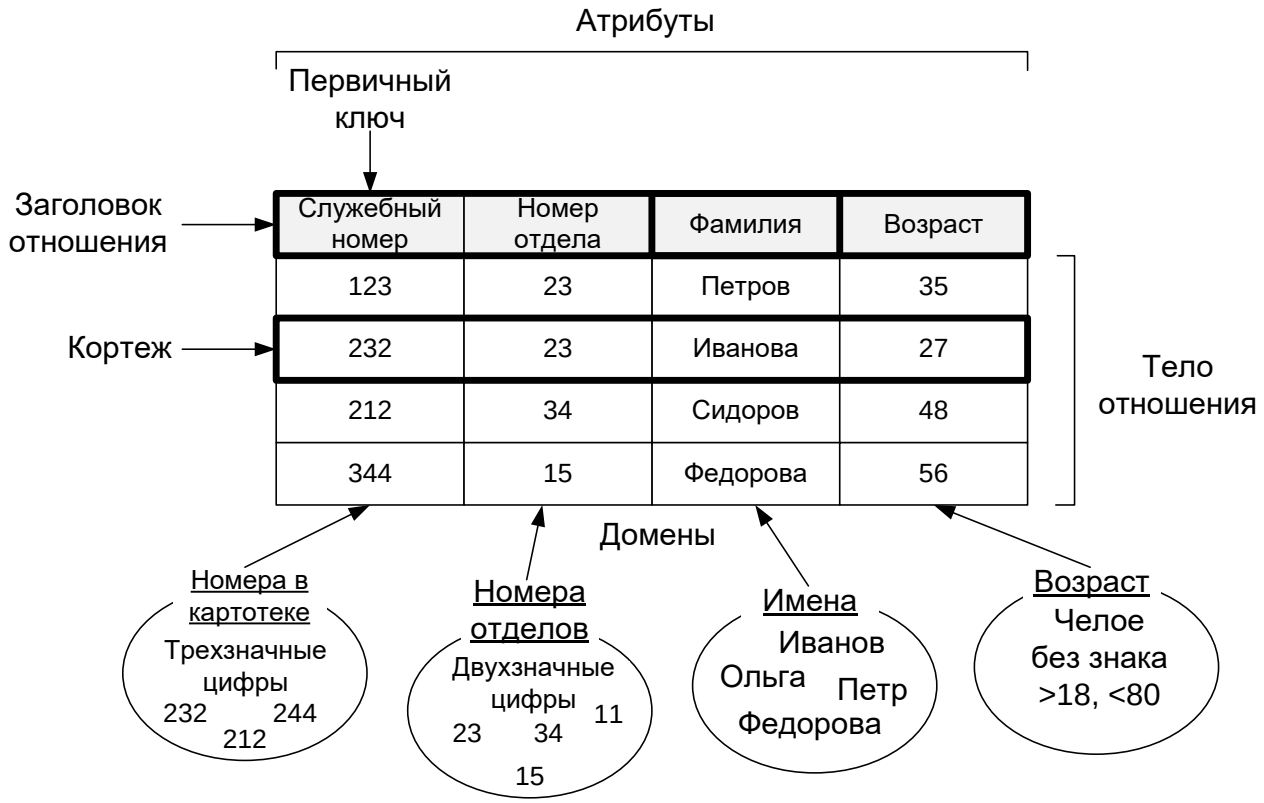


Рис. 3. Основные элементы реляционной модели на примере отношения “Сотрудник”

Тип данных. Понятие типа данных в реляционной модели данных полностью совпадает с аналогичным понятием типа данных в языках программирования, например в Delphi: integer – целое число, string - строка . Обычно в современных базах данных на основе реляционной модели допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как "деньги"), а также данных, адаптированных к специализированным способам их обработки (дата, время, временной интервал, валюта).

Домен. Использование понятия домена означает наличие некоторой ограниченной области объектов, значений, свойств с одинаковыми характеристиками, например домен в ферромагнетиках – область, в которой существует самопроизвольная намагниченность, домен в Интернете – это определенная зона имён, выделенная владельцу домена, например “.ru”, “.com”. С точки зрения моделирования данных домен определяется заданием некоторого базового типа, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Наиболее правильной интуитивной трактовкой понятия домена является его понимание как допустимого потенциального множества значений данного типа. Например, фамилия никогда не может начинаться на цифру, с мягкого знака, со знаков препинания и т.д, а возраст не может быть меньше нуля. Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену, ведь сравнивать возраст и служебный номер бессмысленно, хотя они имеют числовой тип.

Схема отношения, схема базы данных - это именованное множество пар {имя атрибута, имя домена (или типа данных , если понятие домена не поддерживается)}. Степень или "арность" схемы отношения - мощность этого множества. Например, если степень отношения “Сотрудник” равна четырем, то оно является 4-арным. Схема БД - это набор именованных схем отношений.

Кортеж - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым степень, или "арность", кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения, т.е. кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. В классических реляционных базах данных после определения схемы базы данных изменяются только отношения-экземпляры. В них могут появляться новые и удаляться или модифицироваться существующие кортежи. Однако во многих реализациях допускается и изменение схемы базы данных: определение новых и изменение существующих схем отношения (в простой интерпретации это добавление новых столбцов в таблицу). Это принято называть эволюцией схемы базы данных.

Первичный ключ - это атрибут (или набор атрибутов), который может быть использован для однозначной идентификации конкретного кортежа (строки, записи), т.е. в любое время значение первичного ключа в любом кортеже тела отношения отличается от значения первичного ключа в любом другом кортеже тела этого отношения. Первичный ключ не должен иметь дополнительных атрибутов. Это значит, что если из первичного ключа исключить произвольный атрибут, оставшихся атрибутов будет недостаточно для однозначной идентификации отдельных кортежей.

На практике часто в качестве первичного ключа используется искусственный атрибут, называемый идентификатором - ID. Хотя он и не имеет смысла в предметной области, что нарушает реляционную модель, он необходим, чтобы исключить вероятность того, что первичный ключ будет

изменен. Например, если у сотрудника изменится служебный номер и он является первичным ключом, будет необходимо скорректировать все связи отношения “Сотрудник” с другими отношениями. Обычно этот ключ формируется с помощью счетчика, причем его значения никогда не повторяются.

Внешний ключ. Это понятие отсутствует в реляционной модели, поскольку модель не предусматривает каких-либо специальных механизмов связывания отношений, однако широко используется во многих системах управления базами данных как способ поддержания ссылочной целостности данных. Смысл этого механизма состоит в том, что некоему атрибуту (или группе атрибутов) одного отношения назначается ссылка на первичный ключ другого отношения; тем самым закрепляются связи подчиненности между этими отношениями. При этом отношение, на первичный ключ которого ссылается внешний ключ другого отношения, называется master-отношением, или главным отношением; а отношение, от которого исходит ссылка, называется detail-отношением, или подчиненным отношением. Процесс закрепления связи может реализовываться жестко с использованием специфических средств систем управления базами данных (естественно, с нарушением реляционной концепции) или на уровне языка манипулирования данных, в этом случае связь существует только на момент выполнения запроса к базе данных. На рис. 3 атрибут “Номер отдела” может быть использован для связи сотрудника со своим отделом, т.е. связи двух отношений “Сотрудник” и ”Отдел”.

Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных. Принято считать, что реляционный подход к организации баз данных был заложен в конце 1960-х гг. Эдгаром Коддом.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В данной лабораторной работе вы должны получить навыки создания таблиц базы данных средствами среды Delphi. В процессе работы вы должны усвоить основные понятия реляционной модели.

Лабораторная работа проходит в два этапа.

Этап 1. Самостоятельная подготовка к лабораторной работе. Получение допуска преподавателя к выполнению лабораторной работы.

Этап 2. Выполнение лабораторной работы.

Этап 1. Подготовка к лабораторной работе

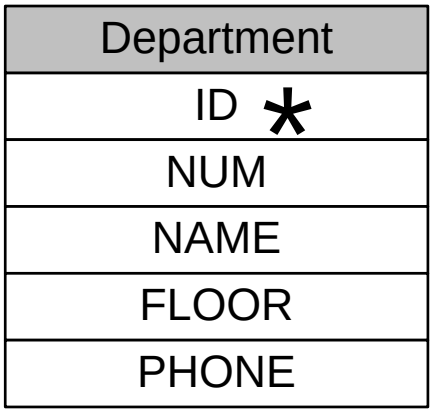
Разработайте схему базы данных, состоящую из двух отношений. Предметная область и соответствующие ей отношения, которые будет описывать база данных, выбираются согласно таблице 1. Вы можете само-

стоятельно предложить преподавателю свой вариант предметной области и название отношений.

Таблица 1

Номер группы	Название предметной области	Отношение (таблица) №1	Отношение (таблица) №2
1	Склад	Поставщик	Товар
2	Связь	Тариф	Абонент
3	Учеба	Студент	Лекция
4	Служба доставки	Курьер	Товар
5	Банк	Клиент	Менеджер
6	Поликлиника	Пациент	Врач
7	Спортклуб	Тренажер	Посетитель
8	Лекции	Лектор	Курс
9	Библиотека	Книга	Абонент
10	Недвижимость	Квартира	Владелец
11	Работа	Задание	Сотрудник
12	Отдел кадров	Сотрудник	Отдел

Для каждого отношения разработайте схему отношения. Схема должна иметь степень не меньше 5. Пример разработанной схемы отношения “Отдел” приведен на рис. 4:



Название отношения: ОТДЕЛ
Атрибут 1: Идентификатор ID - Первичный ключ (уникальный номер каждой записи) – прямой счетчик целых чисел.
Атрибут 2: Номер отдела – Число без знака
Атрибут 3: Название отдела – Текст, длина 50 символов
Атрибут 4: Этаж - Число без знака
Атрибут 5: Телефон – Число без знака (или текст длиной 11 символов или более, если есть разделители)

Рис.4. Разработка схемы отношения “Отдел” (Department)

Для каждого русского названия атрибута должен быть представлен английский вариант в переводе или в транслите. Разработанные схемы отношений приведите в отчете. Данный отчет будет являться допуском к лабораторной работе.

Этап 2. Выполнение лабораторной работы

Для работы с таблицами при проектировании базы данных в среде Delhi предусмотрена специальная утилита Database Desktop, которая позволяет выполнять следующие действия:

- создание таблицы;
- изменение структуры таблиц;
- редактирование записей в таблицах.

Перед запуском этой программы осуществим подготовительные действия: с помощью “Проводника” или любого другого файл-менеджера, доступного на вашем компьютере, создайте в папке вашей группы новую папку с названием DataBase. В этой папке будут храниться все файлы вашей базы данных вне зависимости от номера лабораторной работы.

Запуск Database Desktop можно выполнить двумя способами:

1. С рабочего стола: Пуск -> Borland Delphi - > Database Desktop.
2. Из главного меню Delphi: Tools -> Database Desktop.

После запуска программы разверните ее окно на весь экран.

Процесс создания новой таблицы начните командой из меню File -> New -> Table. В начале создания новой таблицы в окне Create Table (Создание таблицы) выбирается ее формат. По умолчанию предлагается формат таблицы Paradox версии 7, который мы и будем использовать. Нажмите кнопку “ОК”, в результате появится окно определения структуры, в котором вам предлагается создать новую таблицу путем задания ее полей, их типов.

Имя поля в таблице формата Paradox представляет собой строку, написание которой подчиняется следующим правилам:

- имя должно быть не длиннее 25 символов;
- имя не должно начинаться с пробела, вместо пробелов в качестве разделителя слов лучше использовать знак подчеркивания “_”;
- имя не должно содержать квадратные, круглые или фигурные скобки [], () или {}, тире, а также комбинацию символов "тире" и "больше" (->);
- имя не должно быть только символом #, хотя этот символ может присутствовать в имени среди других символов.

Итак, откройте свой отчет и введите предварительно созданные вами поля. Названия полей должны быть только по-английски, переход от одного поля к другому осуществляется с помощью клавиши <Enter>. Тип поля

(число, символ, графика) указывается в колонке Type при нажатии правой клавиши мыши на соответствующую позицию. Список полей, которые вы должны использовать приведен в таблице 2.

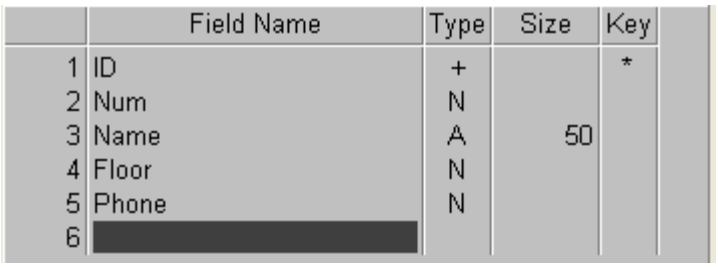
Таблица 2. Рекомендуемые типы полей для данной лабораторной работы

Атрибут отношения	Тип	Описание
Первичный ключ	+ (Autoincrement)	Поле длиной 4 байта, содержащее нередактируемое (read-only) значение типа long integer. Значение этого поля автоматически увеличивается (начиная с 1) с шагом 1
Все текстовые поля	Alpha	Строка длиной 1-255 байт, содержащая любые печатаемые символы
Все цифровые поля	Number	Числовое поле длиной 8 байт, значение которого может быть положительным и отрицательным. Диапазон чисел - от 10-308 до 10308 с 15 значащими цифрами

Первым вводится поле под ключ. Назовите его ID или IDKey. Для того чтобы пометить поле как ключевое, надо щелкнуть в колонке Key левой клавишей мыши, и оно отметится звездочкой.

Для символьных полей указывается число символов, которое будет содержать поле. Добавление к списку полей новой строки выполняется переводом курсора вниз на несуществующую строку, в результате чего эта строка появляется в конце списка. Вставка новой строки между существующими строками с описанием полей выполняется нажатием клавиши <Insert>. Новая строка вставляется перед строкой, в которой расположен курсор. Для удаления строки необходимо установить курсор на эту строку и нажать комбинацию клавиш <Ctrl>+<Delete>.

Пример заполнения таблицы Otdel показан на рис. 5.



	Field Name	Type	Size	Key
1	ID	+		*
2	Num	N		
3	Name	A	50	
4	Floor	N		
5	Phone	N		
6				

Рис.5. Структура таблицы Otdel в DataBase Desktop

Теперь, когда поля введены, надо сохранить таблицу в файле с помощью кнопки “Save As” в созданный вами каталог DataBase. Назовите файл по смыслу вашей таблицы, например Otdel. Если вы закрыли табли-

цу, то ее можно открыть вновь из меню: File - > Open -> Table. Для перехода в режим редактирования свойств таблицы предусмотрена команда Restructure: Table -> Restructure.

После создания таблицы с ней можно связать некоторые свойства. Для таблиц формата Paradox можно задать:

Validity Checks (проверка правильности) - относится к полю записи и определяет минимальное и максимальное значение, а также значение по умолчанию. Кроме того, позволяет задать маску ввода.

Table Lookup (таблица подстановки) - позволяет вводить значение в таблицу, используя уже существующее значение в другой таблице.

Secondary Indexes (вторичные индексы) - позволяют организовать доступ к данным в порядке, отличном от порядка, задаваемого первичным ключом.

Referential Integrity (ссылочная целостность) - позволяет задать связи между таблицами и поддерживать эти связи на уровне ядра. Обычно задается после создания всех таблиц в базе данных

Password Security (парольная защита) - позволяет закрыть таблицу паролем.

Table Language (язык таблицы) - позволяет задать для таблицы языковый драйвер.

Из этого списка определим только некоторые свойства.

Задание вторичных индексов (Secondary Indexes). Индексы необходимы для ускорения доступа к записям таблицы. Для таблиц Paradox индекс также называют вторичным индексом.

Для выполнения операций, связанных с заданием индексов, существует пункт Secondary Indexes (вторичные индексы) комбинированного списка, при этом под списком появляются кнопки Define (Определить) и Modify (Изменить), список индексов и кнопка Erase (Удалить). В списке индексов выводятся имена созданных индексов.

В нашем случае создадим вторичные индексы для всех неключевых полей. Для этого необходимо выполнить действия согласно таблице 3.

Таблица 3

Задание вторичных индексов

Действие	**Описание**
Выбрать из выпадающего списка свойств таблицы (справа) пункт Secondary Indexes	Правая панель окна перейдет в режим вторичных индексов.
Нажать кнопку Define	Появится окно определения вторичных индексов
Указать в секции Fields курсором на первое неключевое поле, нажать копку со стрелкой вправо	Название выбранного поля перейдет в секцию Indexed Field

Повторить предыдущее действие для остальных неключевых полей	В секции Indexed Field окажутся все поля за исключением первичного ключа
Нажать кнопку “ОК”	Появится диалог ввода имени файла
Введите название в виде название “таблицы_idx” (например otdel_idex). Нажмите “ОК”	Файл индексов будет сохранен. Его название появится в правой секции

Окончание табл. 3

Задание языкового драйвера (Table Language). Для каждой таблицы необходимо указать основной национальный язык таблицы. Для этого надо из выпадающего списка свойств таблицы (правая панель) выбрать пункт Table Language**.** Нажать кнопку Define и в появившемся окне, с помощью выпадающего списка выбрать пункт “Pdox ANSI Cyrillic”.

Покажите результат преподавателю. После этого, используя полученные навыки, создайте самостоятельно вторую таблицу из вашего отчета.

Внимание! После выполнения всех операций с настройками свойств нужно сохранять все изменения. Используйте кнопку Save As для каждой таблицы.

Дополнительно вы должны вручную заполнить ваши таблицы, создав в каждой из них несколько (не больше 4) записей. Эти записи вам пригодятся для следующих работ. Для этого надо сделать таблицу текущей (указать курсором на ее заголовок) и выбрать пункт меню Table - > Edit Data (F9). Для перехода от одной записи к другой используйте клавишу <Enter>. Поле ID должно устанавливаться автоматически.

Создание псевдонима базы данных

Для успешного доступа к данным приложение должно обладать информацией о местоположении файлов требуемой базы данных. Самый простой способ заключается в явном задании полного пути к каталогу, в котором хранятся файлы базы данных. Но в случае изменения пути, например при переносе готового приложения на компьютер заказчика, разработчик должен перекомпилировать проект с учетом нового пути.

Для решения такого рода проблем в среде Delphi предусмотрен механизм псевдонимов базы данных, который представляет собой именованную структуру, содержащую путь к файлам и некоторые дополнительные параметры.

В первом приближении можно сказать, что вы просто присваиваете маршруту произвольное имя, которое используется в приложении. Тогда при переносе приложения на другой компьютер достаточно создать и одноименный псевдоним и настроить его на нужный каталог. При этом само приложение не требует переделок, так как оно обращается к псевдониму с одним именем.

Для управления псевдонимами баз данных, настройки стандартных и дополнительных драйверов в составе BDE имеется специальная утилита — BDE Administrator (исполняемый файл BDEADMIN.EXE). Утилиту можно вызвать через кнопку “Пуск” -> Borland Delphi -> BDE Administrator или через панель управления.

После запуска откроется основное окно. В верхней части окна утилиты расположена панель инструментов, кнопки которой используются при работе с конкретным элементом настройки BDE. Рабочая область утилиты BDE Administrator представляет собой двухстраничный блокнот.

Страница Databases содержит иерархическое дерево, в узлах которого расположены установленные в системе на данный момент псевдонимы БД. При выборе какого-либо псевдонима в правой части панели появляются путь к файлам базы данных и перечень параметров драйвера, соответствующего псевдониму, которые можно настраивать вручную.

Страница Configuration используется для настройки параметров драйверов BDE, предназначенных для обеспечения доступа к локальным СУБД и серверам БД. Также здесь определяется системная конфигурация BDE, которая включает параметры числовых форматов, форматов даты и времени. Вся информация на этой странице также структурирована в виде иерархического дерева.

Для создания нового псевдонима требуется выбрать команду New из меню Object. Затем в появившемся простом диалоге задается необходимый драйвер. Выберите “Standart” и нажмите кнопку “ОК”. После выбора драйвера в дереве псевдонимов БД появляется новый узел, для драйвера которого требуется установить необходимые параметры:

Вам предложат изменить название псевдонима. Сделайте это изменение. Например - MyBase.

Теперь надо определить путь до каталога с файлами вашей базы данных. Щелкните мышкой на свойстве PATH в правой части окна, в строке PATH появится кнопка с тремя точками, нажмите ее и укажите путь в появившемся диалоговом окне до вашего каталога DataBase. Сохраните настройки, используя пункт меню Object / Apply.

Теперь вы создали файлы базы данных и зарегистрировали их в системе.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое база данных?
2. Что такое предметная область?
3. Что означает "динамически обновляемая"?
4. Перечислите компоненты модели данных.
5. В каких терминах определяется иерархическая модель?
6. Какие требования предъявляются к элементам иерархической модели?
7. Какие достоинства и недостатки иерархической модели?

8. В каких терминах определяется иерархическая модель? Укажите основное отличие от иерархической на примере с вашими таблицами.
9. Каковы достоинства и недостатки сетевой модели?
10. В каких терминах определяется реляционная модель?
11. Что такое домен, кортеж, отношение? Дайте объяснение на примере с вашими таблицами.
12. Что такое первичный ключ?
13. Что такое внешний ключ?
14. Что произойдет если полю ID присвоить тип Number?
15. Как используется механизм псевдонимов? Нарисуйте функциональную схему реализации доступа к базе данных через псевдонимы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Поисковая система Google.ru, запросы: “основы реляционных баз данных”, “базы данных и Delphi”, “работа в Database Desktop”
2. Основы современных баз данных. С.Д. Кузнецов, информационно-аналитические материалы Центра информационных технологий http://www.citforum.ru/database/osbd/contents.shtml
3. Гофман В.Э., Хомоненко А.Д. Работа с базами данных в Delphi. СПб.: БХВ-Петербург, 2001. 656 с.

Р А Б О Т А № 4

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С ЛОКАЛЬНЫМИ БАЗАМИ ДАННЫХ В СРЕДЕ BORLAND DELPHI. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Цель работы: Изучить основные принципы разработки приложений для работы с базами данных в среде Delphi, получить навыки пользовательского интерфейса.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Локальные и удаленные базы данных. Общие сведения

В зависимости от способа размещения программы, которая использует данные, и самих данных, а также от способа разделения данных между несколькими пользователями различают локальные и удаленные базы данных.

Локальные базы данных реализуются, как правило, на автономном персональном компьютере (сеть не используется). Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место. (Например, ваши рабочие места с установленной средой Delphi.) Данные локальной базы данных (файлы данных) локализованы, т. е. находятся на одном устройстве, в качестве которого может выступать диск компьютера или сетевой диск (диск другого компьютера, работающего в сети). Локальные базы данных не обеспечивают одновременный доступ к информации нескольким пользователям. Для обеспечения разделения данных (доступа к данным) между несколькими пользователями (программами, работающими на одном или разных компьютерах) в локальных базах данных используется метод, получивший название "блокировка файлов". Суть этого метода заключается в том, что пока данные используются одним пользователем, другой пользователь не может работать с этими данными, т. е. данные для него закрыты, заблокированы. Несомненным достоинством локальной базы данных является высокая скорость доступа к информации.

Удаленные базы данных ориентированы на коллективное использование информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети. При разработке таких приложений используются серверы (определение появится далее) баз данных для рабочих групп. Удаленные базы данных строятся по технологии "файл-сервер" или "клиент-сервер". Программа работы с удаленной базой данных состоит из двух частей: клиентской и серверной. Клиентская часть программы работает на компьютере пользователя и обеспечивает взаимодействие с серверной программой посредством запросов, передаваемых на удаленный компьютер (сервер), обеспечивая тем самым доступ к данным. Серверная часть программы, работающая на удаленном компьютере, принимает запросы, выполняет их и пересылает данные клиентской программе. Программа, работающая на удаленном сервере, проектируется так, чтобы обеспечить одновременный доступ к базе данных нескольким пользователям.

2. Реализация работы с локальными и удаленными базами данных в среде Delphi

Процессор баз данных Borland Database Engine (BDE). Любое приложение баз данных имеет в своем составе или использует сторонний механизм доступа к данным, который берет на себя подавляющее большинство стандартных низкоуровневых операций работы с базами данных. Например, любое такое приложение при открытии таблицы базы данных должно выполнить примерно одинаковый набор операций:

- поиск местоположения базы данных;
- поиск таблицы, ее открытие и чтение служебной информации;
- чтение данных в соответствии с форматом хранения данных и т. д.

Очевидно, что если все стандартные функции доступа к данным реализовать в виде специальной программы, сервиса или динамической библиотеки, то это существенно упростит разработку приложений баз данных, которым для выполнения той или иной операции достаточно будет вызвать готовую процедуру.

Взаимодействие приложения, созданного в среде разработки Delphi, и базы данных обеспечивает процессор баз данных Borland Database Engine. Он представляет собой набор динамических библиотек, функции которых позволяют не только обращаться к данным, но и эффективно управлять ими на стороне приложения. Структура процессора баз данных BDE показана на рис. 1.

BDE представляет собой набор динамических библиотек, которые передают запросы на получение или модификацию данных из приложения в нужную базу данных и возвращают результат обработки. В процессе работы библиотеки используют вспомогательные файлы языковой поддержки и информацию о настройках среды. BDE должен устанавливаться на всех компьютерах, на которых выполняются Delphi-приложения, осуществляющие работу с БД.

Приложение через BDE передает запрос к базе данных, а обратно получает требуемые данные. Локальные БД располагаются на том же компьютере, что и работающие с ними приложения. В этом случае информационная система имеет локальную архитектуру (левая часть рис. 1).

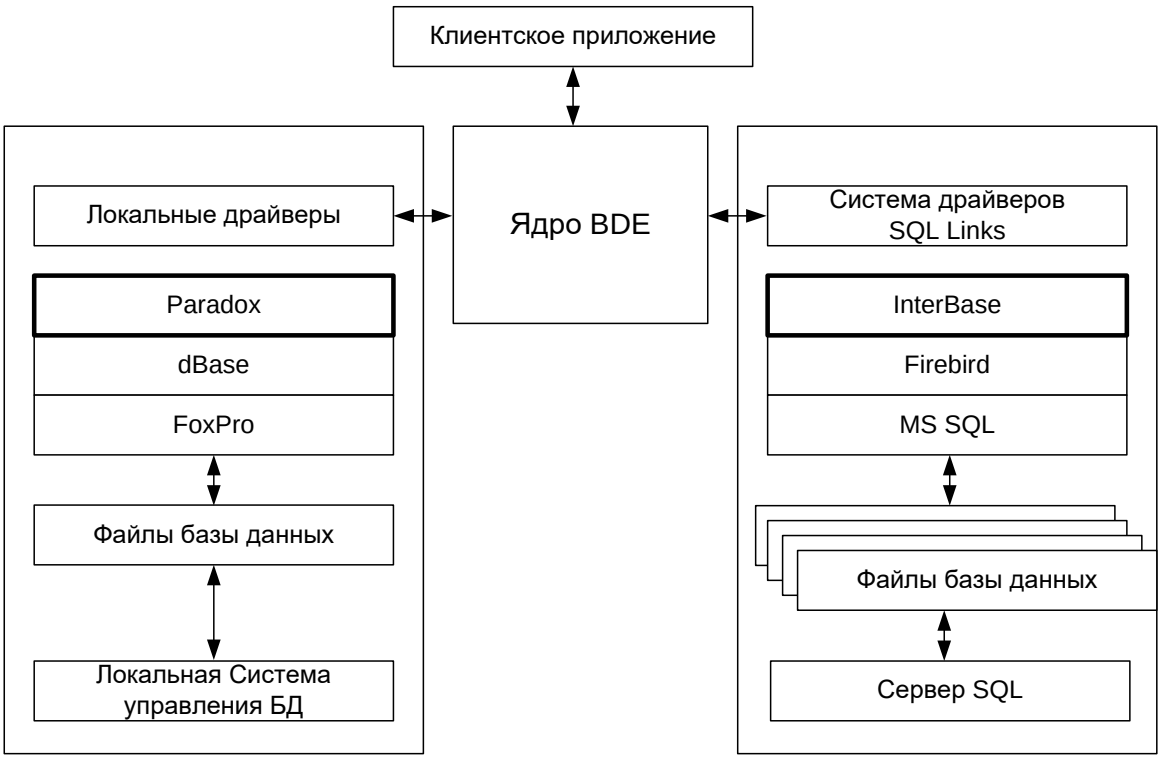


Рис.1. Структура процессора баз данных BDE

В составе BDE поставляются стандартные драйверы, обеспечивающие локальный доступ к СУБД Paradox, dBASE, FoxPro и текстовым файлам. Локальные драйверы устанавливаются автоматически совместно с ядром процессора при установке Delphi. Один из них можно выбрать в качестве стандартного драйвера, который имеет дополнительные настройки, влияющие на функционирование процессора БД.

Доступ к удаленным данным серверов SQL обеспечивает отдельная система драйверов — SQL Links (правая часть рис. 1). С их помощью в Delphi можно без особых проблем разрабатывать приложения для серверов. Эти драйверы необходимо устанавливать дополнительно. Помимо этого, в BDE имеется очень простой механизм подключения любых драйверов ODBC (например, Microsoft Access).

В результате работа с базой данных возможна в трех режимах.

1. ***Локальный режим.*** Клиентское приложении, BDE и файлы базы данных установлены на одном компьютере (рис. 2).

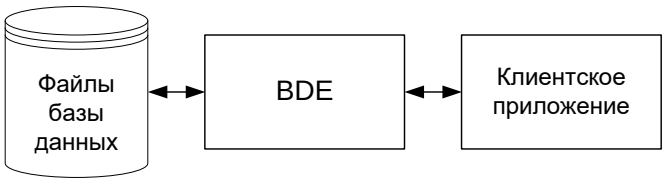


Рис.2. Локальный режим

2. ***Режим файл-сервера.*** При использовании локальной БД в сети возможна организация многопользовательского доступа к ней. В этом случае файлы БД и предназначенное для работы с ней приложение располагаются на сервере сети. Каждый пользователь запускает со своего компьютера это расположенное на сервере приложение, при этом у него запускается копия приложения. Такой сетевой вариант использования локальной БД соответствует архитектуре “файл-сервер” (рис. 3).

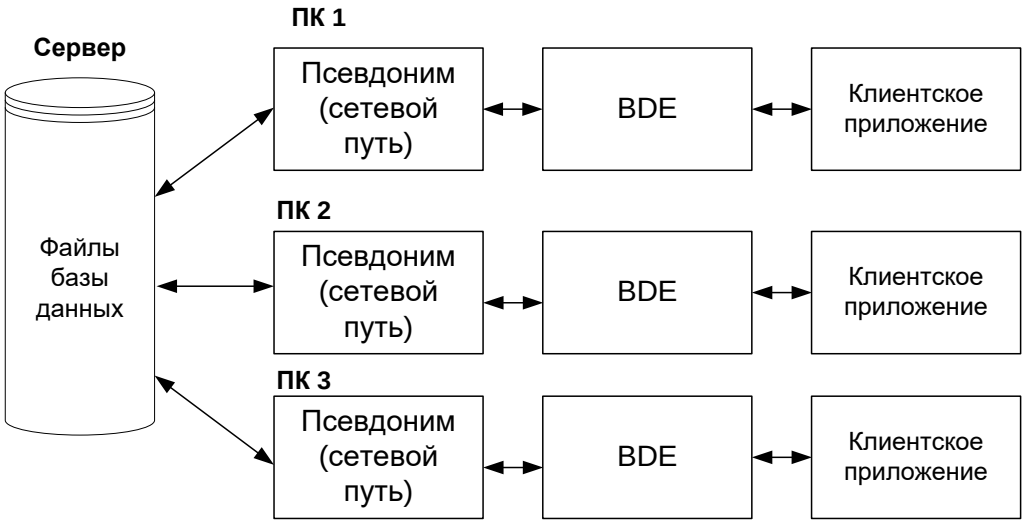


Рис.3. Режим файл-сервера

При такой архитектуре приложение доступа к базе данных должно быть скопировано на каждый компьютер, который находится в локальной сети. Этому приложению должно быть известно местонахождение (путь) файлов базы данных.

Архитектура "файл-сервер" обычно применяется в сетях с небольшим количеством пользователей, для ее реализации подходят персональные СУБД, например Paradox или dBase. Достоинствами этой архитектуры являются простота реализации, а также то, что приложение фактически разрабатывается в расчете на одного пользователя и не зависит от компьютера сети, на который оно устанавливается.

Для успешного доступа к данным приложение и BDE должны обладать информацией о местоположении файлов требуемой базы данных. Задание маршрута входит в обязанности разработчика.

Самый простой способ заключается в явном задании полного пути к каталогу, в котором хранятся файлы базы данных. Но в случае изменения пути, что случается не так уж редко (например, при переносе готового приложения на компьютер заказчика), разработчик должен перекомпилировать проект с учетом будущего местонахождения файлов базы данных или предусмотреть специальные элементы управления, в которых можно задать к ним путь.

Для решения такого рода проблем разработчик может использовать псевдоним (Alias) базы данных, который представляет собой именованную структуру, содержащую путь к файлам БД и некоторые дополнительные параметры. В первом приближении можно сказать, что вы просто присваиваете маршруту произвольное имя, которое используется в приложении. Тогда при переносе приложения на компьютере заказчика достаточно создать стандартными средствами BDE одноименный псевдоним и настроить его на нужный каталог. При этом само приложение не требует переделок, так как оно обращается к псевдониму с одним именем, а вот BDE уже "знает", куда отправить запрос приложения, использовавшего этот псевдоним.

Помимо маршрута к файлам базы данных, псевдоним BDE обязательно содержит информацию о драйвере БД, который используется для доступа к данным. Наличие других параметров зависит от типа драйвера, а значит, от типа СУБД.

Для управления псевдонимами баз данных, настройки стандартных и дополнительных драйверов в составе BDE имеется специальная утилита BDE Administrator (исполняемый файл BDEADMIN.EXE).

3. ***Режим клиент-сервера.*** В архитектуре "клиент-сервер" клиент посылает запрос на предоставление данных и получает только те данные, которые действительно были затребованы. Вся обработка запроса выполняется на удаленном сервере. Приложение при архитектуре "файл-сервер" также может быть записано на каждый компьютер сети, в этом случае

приложению отдельного компьютера должно быть известно местонахождение общей БД (рис. 4), однако приложение не получает непосредственный доступ к файлам базы данных, работаем с ними опосредованно через серверную часть.

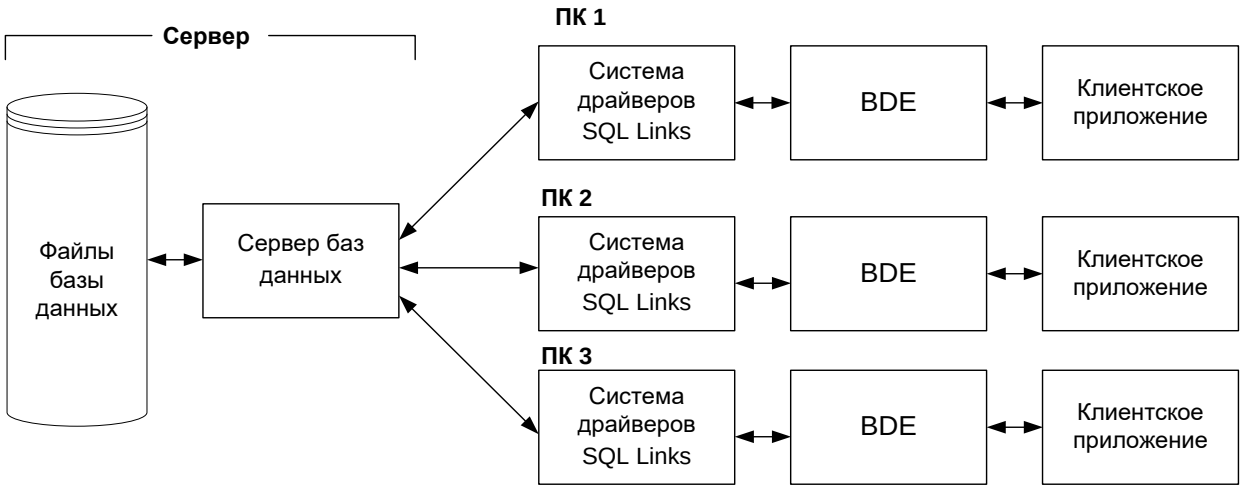


Рис.4 Режим клиент-сервера

3. Общие принципы работы с базами данных в среде Delphi на базе BDE

Разработка приложения баз данных, как и любого другого приложения Delphi, начинается с создания интерфейса — в первую очередь разрабатывается форма. Обычно одна форма отвечает за выполнение группы однородных операций, объединенных общим назначением.

В основе работоспособности любого приложения баз данных лежат наборы данных, которые представляют собой группы записей, переданных из базы данных в приложение для просмотра и редактирования. Каждый набор данных размещается в специальном компоненте доступа к данным. Эти компоненты располагаются на странице Data Access Палитры компонентов.

В приложениях баз данных для работы с набором данных используются компоненты TTable и TQuery, которые обеспечивают создание набора данных для отдельной таблицы БД.

При работе с серверами БД чаще всего используется специальный язык SQL (Structured Query Language — язык структурированных запросов). Для обеспечения использования в приложении наборов данных, созданных на основе запросов SQL, применяется компонент TQuery. В данной работе мы будет работать с компонентом TQuery.

Настройка свойств, отвечающих за подключение к базе данных, в компонентах TTable и TQuery практически не отличается.

Для обеспечения связи набора данных с компонентами отображения данных используется специальный компонент TDataSource. Его роль за-

ключается в управлении потоками данных между набором данных и связанными с ним компонентами отображения данных.

Компоненты отображения данных расположены в закладке **Data Controls.**

Таким образом, для создания работоспособного приложения баз данных необходимо перенести в форму и настроить как минимум три компонента:

- компонент доступа к данным TTable или TQuery;
- компонент TDataSource;
- компонент отображения данных.

Рассмотрим схему взаимодействия этих компонентов в приложении баз данных (рис. 5).

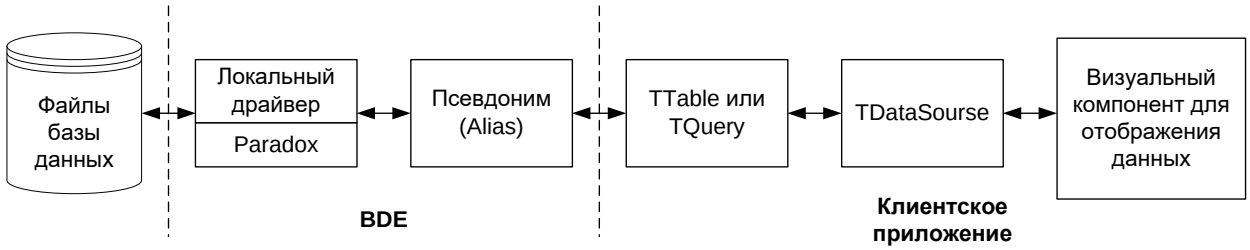


Рис.5. Схема взаимодействия компонентов при разработке приложений локальных баз данных на базе BDE

Непосредственную связь приложения и базы данных осуществляет BDE. Процессор баз данных должен иметь установленный драйвер, через который запросы передаются в БД. Кроме этого, в BDE должен быть зарегистрирован псевдоним, который указывает местоположение файлов БД и тип используемого драйвера

В приложении с BDE взаимодействует компонент доступа к данным (TQuery или TTable), который содержит набор данных и обращается к функциям API BDE для выполнения различных операций. Компонент доступа к данным представляет собой "образ" таблицы базы данных в приложении. Общее число таких компонентов в приложении неограничено.

С каждым компонентом доступа к данным должен быть связан как минимум один компонент TDataSource. В его обязанности входит соединение набора данных с компонентами отображения данных. Он отправляет в эти компоненты текущие значения полей из набора данных и передает обратно сделанные изменения.

Еще одна функция компонента TDataSource заключается в синхронизации поведения компонентов отображения данных с состоянием набора данных. Например, если набор данных не активен, то компонент TDataSource обеспечивает удаление данных из компонентов отображения данных и их перевод в неактивное состояние. Или если набор данных работает в режиме "только для чтения", то компонент TDataSource обязан

передать в компоненты отображения данных запрещение на изменение данных.

С одним компонентом TDataSource могут быть связаны несколько визуальных компонентов отображения данных. Эти компоненты представляют собой модифицированные элементы управления, которые предназначены для показа информации из наборов данных.

При открытии набора данных BDE обеспечивает передачу в набор данных записей из требуемой таблицы БД. Курсор набора данных устанавливается на первую запись. Компонент TDataSource организует передачу в компоненты отображения данных значений необходимых полей из текущей записи.

При перемещении по записям набора данных текущие значения полей в компонентах отображения данных автоматически обновляются.

Пользователь с помощью компонентов отображения данных может просматривать и редактировать данные. Измененные значения сразу же передаются из элемента управления в набор данных с помощью компонента TDataSource. Затем изменения могут быть переданы в базу данных или отменены.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В данной лабораторной работе вы должны получить навыки создания приложений для работы с базами данных в среде Delphi и научиться проектировать пользовательский интерфейс.

Лабораторная работа проходит в два этапа.

Этап 1. Самостоятельная подготовка к лабораторной работе. Получение допуска преподавателя к выполнению лабораторной работы.

Этап 2. Выполнение лабораторной работы.

Этап 1. Подготовка к лабораторной работе

Основные визуальные компоненты, которые помогут вам реализовать поставленные задачи, показаны в таблице 1.

Т

Компоненты для реализации ин

пользователя и доступа к

Компонент	Описание
TForm	Компонент Form – это окно приложения
TQuery	Компонент для работы с набором данных. Набор данных – это совокупность записей, взятых из одной или нескольких таблиц. Содержание набора данных зависит от результатов запросов к вашей базе данных. Компонент находится в закладке BDE
TDataSourse	DataSourse – компонент, который подключает визуальные

	компоненты к набору данных с целью его отображения. Компонент DataSourser является промежуточным звеном между компонентом **TQuery,** соединенным с реальной таблицей БД, и визуальными компонентами. Компонент находится в закладке DataAccess
TDBGrid	Компонент DBGrid обеспечивает табличный способ отображения на экране данных из набора данных TQuery , закладка Data Controls. Этот компонент представляет собой двумерную таблицу, в которой строки представляют собой записи, а столбцы — поля набора данных
TButton	Кнопка, закладка Standard
TLabel	Нередактируемая текстовая надпись, закладка Standard
TEdit	Ввод и отображение текстовой строки, закладка Standard
TPanel	Панель – элемент дизайна графического интерфейса, закладка Standard
TSpeedButton	Кнопка для инструментальных панелей (панелей быстрого запуска), закладка Additional.

Подготовьте бланк отчета о проведении лабораторной работы. В пункте Этап1 приведите описание основных свойств компонентов TQuery, TDataSourser, TDBGrid. Используйте библиографический список.

Этап 2. Выполнение лабораторной работы

Определим задачи, которые должен решать пользовательский интерфейс программы на данном этапе.

1. Просмотр содержимого базы данных.
2. Добавление новых данных (для двух таблиц).
3. Редактирование данных (для двух таблиц).

Данный перечень задач определяет проектируемый интерфейс пользователя, который будет состоять из нескольких окон, которые вам предстоит спроектировать.

Итак, создайте новый проект File ->New->Application. Разместите на появившейся форме компоненты согласно Таблице 2.

Таблица 2

Название компонента	**Описание**	**Закладка**	**Внешний вид**
Panel1	Панель (элемент дизайна)	Samples	
DBGrid	Таблица данных	Data Control	
TQuery	Набор данных	BDE	

Окончание табл. 2

Разместите два набора данных.			
TDataSourser	Источник данных	DataAccess	
Разместите два источника данных.			
Перечисленные ниже компоненты должны размещаться на компоненте Panel1			
TSpeedBut-ton	Кнопка быстрого запуска	Additional	
Разместите четыре кнопки подряд.			

В итоге форма должна принять вид показанный на рис. 6.

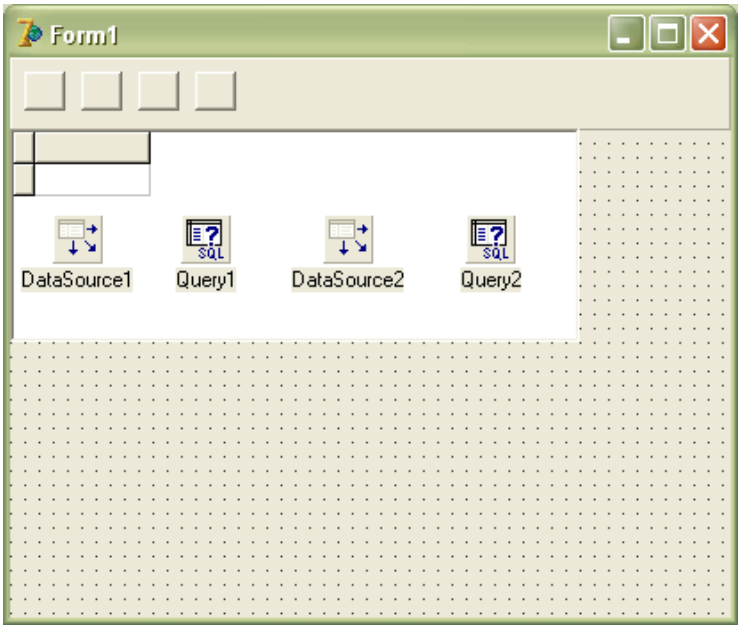


Рис.6. Форма с размещенными компонентами

Через компонент Query1 мы будем отображать данные в главном окне на DBGrid1, а через компонент Query2 мы будет добавлять, редактировать и удалять данные.

После того как все компоненты будут размещены на форме, необходимо повторно сохранить проект. Запустите проект на компиляцию и выполнение. Если все выполнено правильно, произойдет запуск вашей программы.

Пока функциональность компонентов не описана они ничего не “умеют“ делать: выпадающие списки пустые, кнопка не реагирует на нажатие. Закройте программу с помощью стандартных кнопок. Выполните действия согласно таблице 3. При программировании реакции на событие переход в окно исходного текста осуществляется двойным щелчком на выбранном событии.

Сохраните форму и связанный с ней исходный текст командой File – Save или Save As под именем, например, Main.

Таблица 3

Выделенный объект (на форме)	Вкладка окна Object Inspector (F11)	Имя свойства/ имя события	Действие
Form1	Properties	Caption	Установка имени формы «Локальная база данных»
Panel	Properties	Align	Установка режима выравнивания по верхней кромке: alTop
DBGrid1	Properties	Align	Установка режима выравнивания по всей поверхности: alClient
	Properties	DataSource	Из выпадающего списка выбрать DataSourser1
Query1	Properties	Database-Name	Указатель на псевдоним вашей базы данных. Выберите ваш псевдоним (созданный в предыдущей работе)
	Properties	SQL	Нажать … . В появившемся окне с верхней строки написать: Select * from имя_ файла_ первой_ таблицы. Нажать “ОК”.
	Properties	DataSource	Из выпадающего списка выбрать DataSourser1
Query2	Properties	Database-Name	Указатель на псевдоним вашей базы данных. Выберете ваш псевдоним (созданный в предыдущей работе)
	Properties	DataSource	Из выпадающего списка выбрать DataSourser2
Четыре кнопки вызывают: 2 окна добавления новой записи в таблицу 1 и в таблицу 2, 2 окна редактирования записи таблицы 1 и таблицы 2			
SpeedButton1	Properties	Flat	Режим рисования: True - плоская кнопка
	Properties	Glyph	Иконка на кнопке. Нажать … . В появившемся окне нажать кнопку Load. Перейди в каталог C:\Program Files\Borland\Delphi7\Borland Shared\Images\Buttons Найти подходящую картинку. Нажать “Открыть”. Нажать “ОК”
Выполните данные действия для остальных трех кнопок			

Проверка правильности работы: установите свойство Active компонента Query1 как True. Если ошибок нет, то в компоненте DBGrid появится содержимое первой таблицы. Покажите результат преподавателю. Не продолжайте выполнение, если возникла ошибка – исправьте ее.

Для добавления новой записи в таблицу 1 и таблицу 2 создадим два окна. Далее будет описываться пример создания только одного окна, для другого – разработка производится по аналогии .

Итак, создайте новую форму. File ->New->Form. Разместите на появившейся форме компоненты, согласно Таблице 4.

Таблица 4.

Название компонента	**Описание**	**Закладка**	**Внешний вид**
Panel1	Панель (элемент дизайна)	Samples	Panel1
Button1	Кнопка	Standard	Button1
Button2	Кнопка	Standard	Button1
Перечисленные ниже компоненты должны размещаться на компоненте Panel1 парами по числу столбцов (атрибутов) вашей таблицы			
Label1	Текстовая метка	Standard	Label1
Edit1	Строка редактирования	Standard	Edit1

В итоге форма должна принять вид показанный на рис. 7 (пример для таблицы “Отдел”).

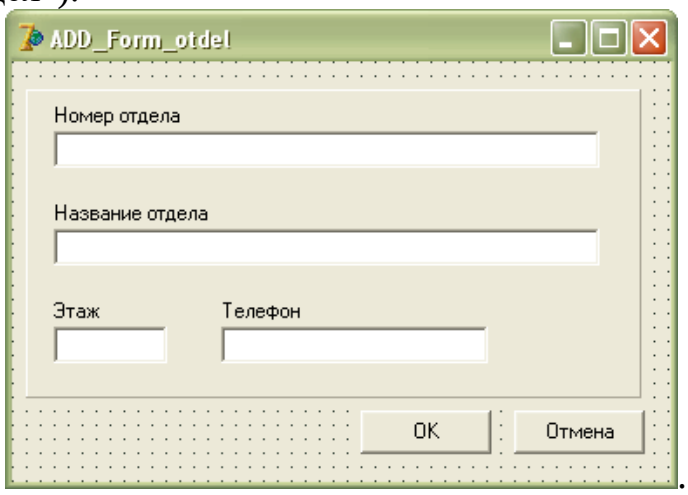


Рис.7. Внешний вид формы добавления новой записи об отделе

Сохраните форму и связанный с ней исходный текст командой File – Save.

Выполните действия согласно таблице 5. При программировании реакции на событие переход в окно исходного текста осуществляется двойным щелчком на выбранном событии.

Таблица 5

Выделенный объект (на форме)	Вкладка окна Object Inspector (F11)	Имя свойства/ имя события	Действие
Form1	Properties	Caption	Установка имени формы «Добавление нового отдела»
	Properties	BorderStyle	Измените свойство режима вывода окна bsDialog
Panel	Properties	Align	Установка режима выравнивания по верхней кромке: alTop
Button1	Properties	Caption	Установка имени кнопки “OK”
Button2	Properties	Caption	Установка имени кнопки “Отмена”
	Events	OnClick	Событие при нажатии на кнопку. Исходный код: Close;

Повторно сохраните форму и связанный с ней исходный текст командой File – Save или Save As.

Теперь необходимо сообщить главной форме Form1 о существовании вновь созданной формы Form2 и наоборот. Для этого нужно перейди в окно исходного текста формы Form1 (выбрать закладку unit1 в окне исходного текста). В тексте модуля unit1 найти секцию **implementation** и после нее добавить информацию о новой форме с помощью оператора uses, написать: **uses unit2;.** Аналогично в исходном тексте формы добавления Form2 (закладка unit2) напишите: uses unit1;.

Сохраните (File – Save All) и скомпилируйте и запустите проект "Run" (F9) – ошибок быть не должно.

Теперь реализуем функцию вызова спроектированного диалогового окна в режиме выполнения работы программы.

Примечание: диалоговое окно вызывается методом ShowModal (В терминологии Windows модальными окнами называются такие специальные окна, которые, раз появившись на экране, блокируют работу пользователя с другими окнами вплоть до своего закрытия.)

Перейдите на главную форму Form1 и выполните действия согласно Таблице 6.

Таблица 6

Выделенный объект (на форме)	Вкладка окна Object Inspector (F11)	Имя свойства/ имя события	Действие
SpeedButton1	Events	OnClick	Событие при нажатии на кнопку добавления данных первой таблицы. Исходный код: Form2.ShowModal;

Повторно сохраните (File -> Save All), скомпилируйте и запустите проект"Run" (F9) – ошибок быть не должно. Щелкните на первой кнопке, и появится окно Добавление отдела.

Аналогичным образом создайте еще 3 окна. Окна редактирования будут повторять окна добавления – разница пока что будет в названии Caption соответствующего окна. Выполните аналогичные действия для других кнопок быстрого запуска, вписывая в их функции обработки щелчка мышью команду Название__окна.ShowModal. После каждого добавления проверяйте правильность компиляции и запуска. У вас должны открываться все созданные вами диалоговые окна. Покажите результат преподавателю. Наполнение функциональностью созданного интерфейса будет происходить в следующих лабораторных работах.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличаются локальные и удаленные информационные системы? Какова их область применения?
2. Что такое одноранговые сети и сети с выделенным сервером?
3. Что такое технология файл-сервер? Как она реализуется? Перечислите ее достоинства и недостатки.
4. Приведите пример реализации технологии файл-сервер в среде Delphi.
5. Что такое технология клиент-сервер? Как она реализуется? Перечислите ее достоинства и недостатки.
6. Что такое модуль BDE? Какова его структура?
7. Пример реализации технологии клиент-сервер в среде Delphi. Функциональная схема. Перечень и назначение используемых компонентов.
8. Компонент TQuery. Назначение, основные свойства, функциональная схема использования.
9. Компонент TDataSourser. Назначение, основные свойства, функциональная схема использования.
10. Компонент TDBGrid. Назначение, основные свойства, функциональная схема использования.
11. Что такое модальное окно?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Поисковая система Google.ru, запросы: “что такое BDE”, “компонент TQuery ”, “компонент TDBGrid”, “технология файл-сервер”, “технология клиент-сервер ”
2. Гофман В.Э., Хомоненко А.Д. Работа с базами данных в Delphi. СПб.: БХВ-Петербург, 2001. 656 с.

С О Д Е Р Ж А Н И Е

Разработка приложений доступа к базам данных
в среде Borland Delphi. Часть 1.

Составители А л п а т о в Алексей Викторович
Г у д з е в Валерий Владимирович
М е л ь н и к Ольга Владимировна

Редактор Р.К. Мангутова
Корректор С.В. Макушина
Подписано в печать 29.06.09. Формат бумаги 60/84 1/16.
Бумага газетная. Печать трафаретная. Усл. печ. л. 3,75.
Уч.–изд. л. 3,75. Тираж 30 экз. Заказ
Рязанский государственный радиотехнический университет.
391000, Рязань, ул. Гагарина, 59/1.
Редакционно–издательский центр РГРТУ.