Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический университет имени П. О. Сухого»

Институт повышения квалификации и переподготовки кадров

Кафедра «Информатика»

А. И. Рябченко, Н. В. Самовендюк, Н. С. Емельянченко

СРЕДСТВА ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по одноименной дисциплине
для слушателей специальности 1-40 01 73
«Программное обеспечение информационных систем»
заочной формы обучения

Гомель 2013

УДК 004.421+004.43(075.8)
ББК 32.973.26-018.1я73
Р98

Рекомендовано кафедрой «Информатика» ГГТУ им. П. О. Сухого
(*протокол № 5 от 27.11.2012 г.*)

Рецензент: д-р техн. наук, проф. каф. «Информационные технологии»
ГГТУ им. П. О. Сухого *И. А. Мурашко*

Рябченко, А. И.

Р98 Средства визуального программирования приложений : лаборатор. практикум по одноим. дисциплине для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / А. И. Рябченко, Н. В. Самовендюк, Н. С. Емельянченко. – Гомель : ГГТУ им. П. О. Сухого, 2013. – 60 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://library.gstu.by/StartEK/. – Загл. с титул. экрана.

Представлены краткие теоретические сведения и задания к лабораторным работам, направленные на разработку приложений с использованием библиотеки визуальных компонентов Delphi.

Приводится обзор визуальных компонент, применяемых при разработке типовых приложений в ОС Windows. Большое внимание уделено разработке программ для работы с локальными и сетевыми базами данных с использованием технологий BDE и ADO. Описываются возможности создания визуальных приложений, способных формировать предварительный просмотр и печать данных, использовать COM-технологии для передачи данных в стандартные приложения пакета MS Office.

Для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения.

УДК 004.421+004.43(075.8)
ББК 32.973.26-018.1я73

Лабораторная работа № 1

Тема: Использование визуальных компонент при разработке приложений

Цель работы: ознакомиться с визуальными компонентами, применяемыми при разработке типовых приложений в ОС Windows.

Теоретические сведения

В Delphi имеется два компонента, представляющие меню: MainMenu — главное меню, и PopupMenu — всплывающее меню. Оба компонента расположены на странице "Standard". Эти компоненты имеют много общего.

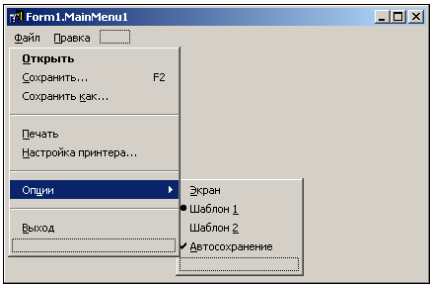
MainMenu

Это невизуальный компонент, т.е. место его размещения на форме в процессе проектирования не имеет никакого значения для пользователя — он все равно увидит не сам компонент, а только меню, сгенерированное им.

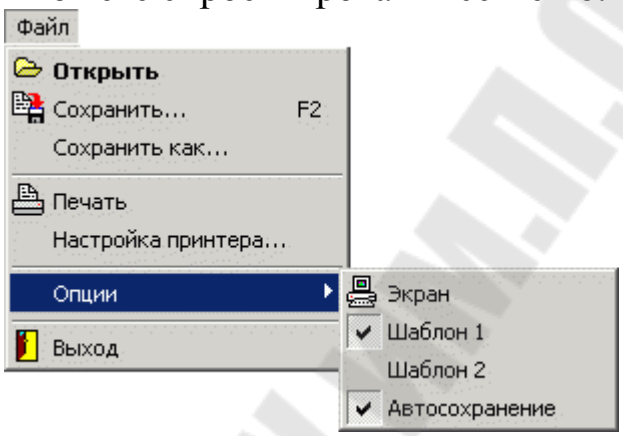
Обычно на форму помещается один компонент MainMenu. В этом случае его имя автоматически заносится в свойство формы Menu. Но можно поместить на форму и несколько компонентов MainMenu с разными наборами разделов, соответствующими различным режимам работы приложения. В этом случае во время проектирования свойству Menu формы присваивается ссылка на один из этих компонентов. А в процессе выполнения в нужные моменты это свойство можно изменять, меняя соответственно состав главного меню приложения.

Основное свойство компонента — Items. Его заполнение производится с помощью Конструктора Меню, вызываемого двойным щелчком на компоненте MainMenu или нажатием кнопки с многоточием рядом со свойством Items в окне Инспектора Объектов.

В результате откроется окно.



В этом окне вы можете спроектировать все меню.



При работе в конструкторе меню новые разделы можно вводить, помещая курсор в рамку из точек, обозначающую место расположения нового раздела. Если при этом раздел ввелся не на нужном вам месте, вы можете отбуксировать его мышью туда, куда вам надо. Другой путь ввода нового раздела — использование контекстного меню, всплывающего при щелчке правой кнопкой мыши. Если вы предварительно выделите какой-то раздел меню и выберите из контекстного меню команду "Insert", то рамка нового раздела вставится перед ранее выделенным. Из контекстного меню вы можете также выполнить команду "Create Submenu", позволяющую ввести подменю в выделенный раздел (см. подменю раздела "Опции").

При выборе нового раздела вы увидите в Инспекторе Объектов множество свойств данного раздела. Дело в том, что каждый раздел меню, т.е. каждый элемент свойства Items, является объектом типа TMenuItem, обладающим своими свойствами, методами, событиями.

Свойство **Caption** обозначает надпись раздела. Заполнение этого свойства подчиняется тем же правилам, что и заполнение аналогичного свойства в кнопках, включая использование символа амперсанта для обозначения клавиш быстрого доступа. Если вы в

качестве значения Caption очередного раздела введете символ минус «-», то вместо раздела в меню появится разделитель.

Разделители после разделов "Сохранить как", "Настройка принтера" и "Опции").

Свойство **Name** задает имя объекта, соответствующего разделу меню. Очень полезно давать этим объектам осмысленные имена, так как иначе вы скоро запутаетесь в ничего не говорящих именах типа "N21". Куда понятнее имена типа "MFile", "MOpen", "MSave" и т.п.

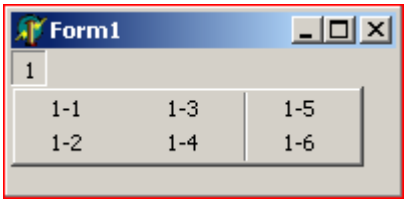
Свойство **Shortcut** определяет клавиши быстрого доступа к разделу меню — «горячие» клавиши, с помощью которых пользователь, даже не заходя в меню, может в любой момент вызвать выполнение процедуры, связанной с данным разделом. Чтобы определить клавиши быстрого доступа, надо открыть выпадающий список свойства Shortcut в окне Инспектора Объектов и выбрать из него нужную комбинацию клавиш. Эта комбинация появится в строке раздела меню.

Свойство **Default** определяет, является ли данный раздел разделом по умолчанию своего подменю, т.е. разделом, выполняемым при двойном щелчке пользователя на родительском разделе. Подменю может содержать только один раздел по умолчанию, выделяемый жирным шрифтом (см. раздел "Открыть").

Свойство **Break** используется в длинных меню, чтобы разбить список разделов н а несколько столбцов. Возможные значение **Break**:

mbNone — отсутствие разбиения меню (это значение принято по умолчанию), mbBarBreak и mbBreak — в меню вводится новый столбец разделов, отделенный от предыдущего полосой (mbBarBreak) или пробелами (mbBreak).

Пример, в котором в разделе 1-3 установлено значение Break = mbBreak, а в разделе 1-5 — Break = mbBarBreak.



Свойство **Checked**, установленное в true, указывает, что в разделе меню будет отображаться маркер флажка, показывающий, что данный раздел выбран (раздел «"Автосохранение"»). Правда, сам по себе этот маркер не изменяется и в обработчик события OnClick такого раздела надо вставлять оператор типа

```
MAutoSave.Checked := not MAutoSave.Checked;
```

(в приведенном операторе подразумевается, что раздел меню назван MAutoSave).

Еще одним свойством, позволяющим вводить маркеры в разделы меню, является **RadioItem**. Это свойство, установленное в true, определяет, что данный раздел должен работать в режиме радиокнопки совместно с другими разделами, имеющими то же значение свойства **GroupIndex**.

По умолчанию значение **GroupIndex** равно 0. Но можно задать его большим нуля и тогда, если имеется несколько разделов с одинаковым значением **GroupIndex** и с **RadioItem** = **true**, то в них могут появляться маркеры флажков, причем только в одном из них (свойство **RadioItem** установлено в **true** в разделах "Шаблон 1" и "Шаблон 2", имеющих одинаковое значение **GroupIndex**). Если вы зададите программно в одном из этих разделов Checked = true, то в остальных разделах **Checked** автоматически сбросится в false. Впрочем, установка **Checked** = **true** лежит на программе; эта установка может выполняться аналогично приведенному выше оператору.

Описанные маркеры флажков в режиме радиокнопок и в обычном режиме используются для разделов меню, представляющих собой различные опции, взаимоисключающие или совместимые.

Для каждого раздела могут быть установлены во время проектирования или программно во время выполнения свойства **Enabled** (доступен) и **Visible** (видимый).

Если установить **Enabled** = **false**, то раздел будет изображаться серой надписью и не будет реагировать на щелчок пользователя. Если же задать **Visible** = **false**, то раздел вообще не будет виден, а остальные разделы сомкнутся, заняв место невидимого. Свойства **Enabled** и **Visible** используются для того, чтобы изменять состав доступных пользователю разделов в зависимости от режима работы приложения.

В Delphi предусмотрена возможность ввода в разделы меню изображений. За это ответственны свойства разделов **Bitmap** и **ImageIndex**. Первое из них позволяет непосредственно ввести изображение в раздел, выбрав его из указанного вами файла. Второе позволяет указать индекс изображения, хранящегося во внешнем компоненте **ImageList**. Указание на этот компонент вы можете задать в свойстве Images компонента **MainMenu**. Индексы начинаются с 0.

Если вы укажете индекс -1 (значение по умолчанию), изображения не будет.

Основное событие — OnClick, возникающее при щелчке пользователя на разделе или при нажатии «горячих» клавиш быстрого доступа.

Контекстное всплывающее меню — компонент PopupMenu

Контекстное меню привязано к конкретным компонентам. Оно всплывает, если во время, когда данный компонент в фокусе, пользователь щелкнет правой кнопкой мыши. Обычно в контекстное меню включают те команды главного меню, которые в первую очередь могут потребоваться при работе с данным компонентом.

Контекстному меню соответствует компонент PopupMenu. Поскольку в приложении может быть несколько контекстных меню, то и компонентов PopupMenu может быть несколько. Оконные компоненты: панели, окна редактирования, а также метки и др. имеют свойство PopupMenu, которое по умолчанию пусто, но куда можно поместить имя того компонента PopupMenu, с которым будет связан данный компонент.

Формирование контекстного всплывающего меню производится с помощью Конструктора Меню, вызываемого двойным щелчком на PopupMenu, точно так же, как это делалось для главного меню. Обратим только внимание на возможность упрощения этой работы. Поскольку разделы контекстного меню обычно повторяют некоторые разделы уже сформированного главного меню, то можно обойтись копированием соответствующих разделов.

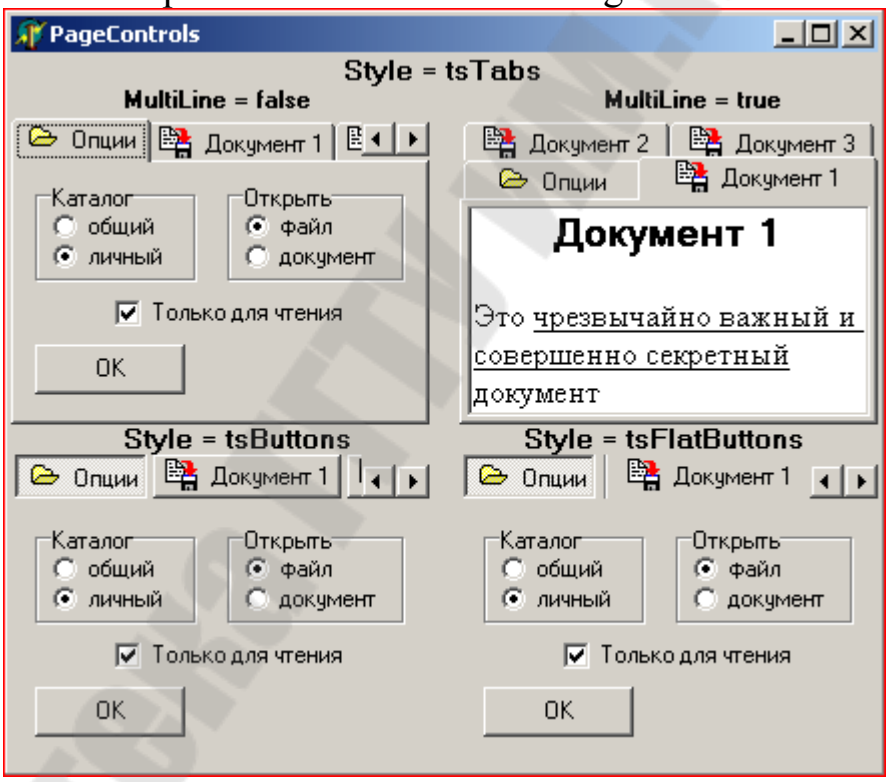
Для этого, войдя в Конструктор Меню из компонента PopupMenu, щелкните правой кнопкой мыши и из всплывшего меню выберите команду "Select Menu" (выбрать меню). Вам будет предложено диалоговое окно, в котором вы можете перейти в главное меню. В нем вы можете выделить нужный вам раздел или разделы (при нажатой клавише Shift выделяются разделы в заданном диапазоне, при нажатой клавише Ctrl можно выделить совокупность разделов, не являющихся соседними). Затем выполните копирование их в буфер обмена, нажав клавиши Ctrl-C. После этого опять щелкните правой кнопкой мыши, выберите команду "Select Menu" и вернитесь в контекстное меню. Укажите курсором место, в которое хотите вставить скопированные разделы, и нажмите клавиши чтения

из буфера обмена — Ctrl-V. Разделы меню вместе со всеми их свойствами будут скопированы в создаваемое вами контекстное меню.

В остальном работа с PopupMenu не отличается от работы с MainMenu. Только не возникает вопросов объединения меню разных форм: контекстные меню не объединяются.

Многостраничные панели

Многостраничные панели позволяют экономить пространство окна приложения, размещая на одном и том же месте страницы разного содержания. На рисунке показаны различные формы отображения многостраничного компонента PageControl.



Чтобы задавать и редактировать страницы этого компонента, надо щелкнуть на нем правой кнопкой мыши. Во всплывшем меню вы можете видеть команды: "New Page" — создать новую страницу, "Next Page" — переключиться на следующую страницу, "Previous Page" — переключиться на предыдущую страницу.

Каждая создаваемая страница является объектом типа **TTabSheet**. Это панель, на которой можно размещать любые управляющие компоненты, окна редактирования и т.п. После того,

как вы создадите несколько страниц, выделите одну из них, щелкнув в ее середине, и посмотрите ее свойства в Инспекторе Объектов.

Страница имеет следующие основные свойства:

Name Имя, по которому можно ссылаться на страницу

Caption Надпись, которая появляется на ярлычке закладки

PageIndex Индекс страницы, по которому можно ссылаться на страницу

ImageIndex Индекс изображения, которое может появляться на ярлычке закладки

Из общих свойств компонента **PageControl** можно отметить:

Style Определяет стиль отображения компонента:

tsTabs — закладки (верхние компоненты),

tsButtons — кнопки (левый нижний компонент),

tsFlatButtons — плоские кнопки (правый нижний компонент)

MultiLine Определяет, будут ли закладки размещаться в несколько рядов, если все они не помещаются в один ряд (два одинаковых компонента, но в левом MultiLine = false, а в правом — true; примером компонента с MultiLine = false является также знакомая вам палитра компонентов в Delphi)

TabPosition Определяет место расположения ярлычков закладок: tpBottom — внизу, tpLeft — слева, tpRight — справа и tpTop — вверху компонента (это значение по умолчанию и именно оно задано)

TabHeight и **TabWidth** Высота и ширина ярлычков закладок в пикселях. Если значения этих параметров заданы равными 0, то размеры ярлычков определяются автоматически по размерам надписей на них

Images Ссылка на компонент ImageList (см. раздел 9.3), который содержит список изображений на ярлычках. Свойства ImageIndex страниц содержат индексы, соответствующие именно этому списку

ScrollOpposite Определят способ перемещения закладок при размещении их в несколько рядов (опробуйте экспериментально, как это свойство влияет на поведение компонента)

ActivePage Имя активной страницы

Pages[Index: Integer] Доступ к странице по индексу (первая страница имеет индекс 0). Свойство только для чтения

PageCount Количество страниц. Свойство только для чтения

В компоненте имеется ряд методов, позволяющих оперировать страницами, создавать их, уничтожать, переключать. Посмотрите их во встроенной справке Delphi.

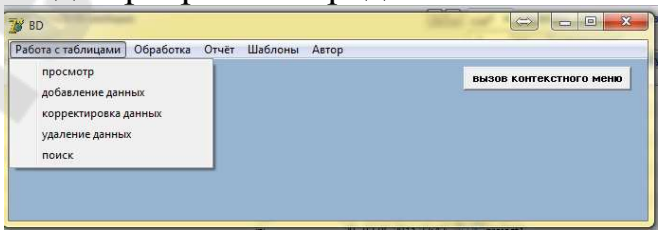
Основные события компонента — **OnChanging** и **OnChange**. Первое из них происходит непосредственно перед переключением на другую страницу после щелчка пользователя на новой закладке. При этом в обработчик события передается по ссылке параметр **AllowChange** — разрешение переключения. Если в обработчике задать **AllowChange = false**, то переключение не произойдет. Событие **OnChange** присходит сразу после переключения.

Задание

Для заданной предметной области разработать приложение, имеющее основные типичные элементы (меню, контекстное меню, информацию о разработчике и др.).

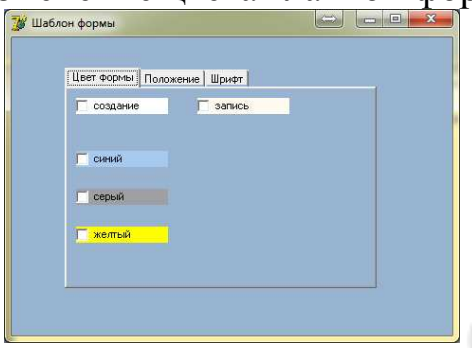
Пояснения к выполнению работы

1. Изучить свойства заданных компонент.
2. Создать проект, форму и модуль со следующими именами <ЛР1_имя>.dpr, < ЛР1_имя >.pas и < ЛР1_имя >.dfm.
3. Вызвать конструктор меню и создать главное меню с произвольным набором разделов и подменю. Меню должно обеспечивать возможность работы с заданной предметной областью. Образец внешнего вида программы представлен ниже.

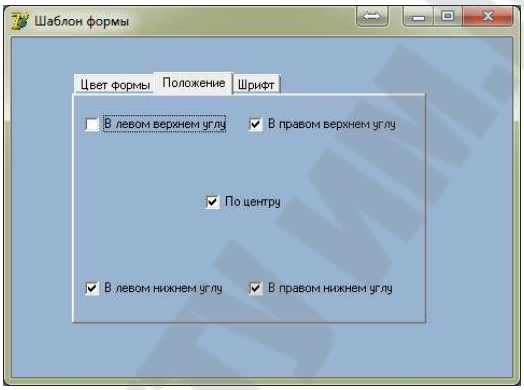


4. Аналогичным образом создать всплывающее контекстное меню PopupMenu, дублирующее основные действия главного меню.
5. Создать дополнительную форму, разместив на ней компоненту TabSheet. Сделать три закладки. На каждой из закладок

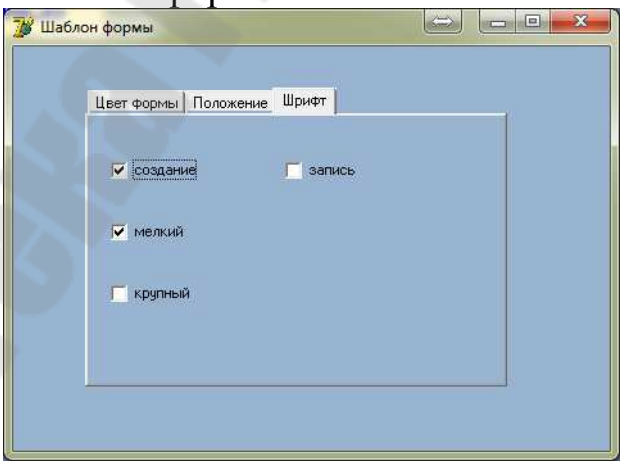
разместить компоненты CheckBox. Компоненты первой закладки должны обеспечивать изменение цвета главной формы приложения.



Компоненты второй закладки должны изменять положение главной формы на экране.



Компоненты третьей закладки должны изменять размер и начертание шрифта главной формы.



6. В программе предусмотреть возможность сохранения положения, цвета, размера и начертания шрифта главного окна. Для этого можно воспользоваться возможностями класса TIniFile.

Следующий фрагмент кода демонстрирует, как создать IniFile в том же каталоге, где расположена запускаемая программа.

```
procedure TForm2.CheckBox2Click(Sender: TObject);//запись
var
Ini: Tinifile;
begin
IniFile:=TIniFile.Create(extractfilepath(paramstr(0))+'project.ini');
IniFile.WriteInteger('FORM1','Form1Color',Form1.Color);
IniFile.Free;
end;
```

Следующий фрагмент кода показывает, как прочитать данные из IniFile.

```
procedure TForm2.CheckBox4Click(Sender: TObject);
var
Ini: Tinifile;
begin
 //открываем файл
Ini:=TiniFile.Create(extractfilepath(paramstr(0))+ 'MyIni1.ini');
Form1.Width:=Ini.ReadInteger('Size','Width',100);
//последнее значение (100) это значение по умолчанию (default)
Form1.Height:=Ini.ReadInteger('Size','Height',100) ;
Form1.Left:=Ini.ReadInteger('Position','X',10);
Form1.Top:=Ini.ReadInteger('Position','Y',10);
Ini.Free;
end;
```

Контрольные вопросы

1. Свойства и методы компоненты TMainMenu.
2. Свойства и методы компоненты TPopupMenu.
3. Свойства и методы компоненты TTabSheet.
4. Работа с классом TIniFile.

Лабораторная работа № 2

Тема: Создание таблиц с помощью Database Desktop

Цель работы: ознакомиться с процессом создания и редактирования таблиц с помощью утилиты Database Desktop.

Теоретические сведения

Архитектура и функции BDE

Мощность и гибкость Delphi при работе с базами данных основана на низкоуровневом ядре - процессоре баз данных Borland Database Engine (BDE).

BDE позволяет осуществлять доступ к данным как с использованием традиционного record-ориентированного (навигационного) подхода, так и с использованием set-ориентированного подхода, используемого в SQL-серверах баз данных.

Все инструментальные средства баз данных Borland - Paradox, dBase, Database Desktop - используют BDE. Все особенности, имеющиеся в Paradox или dBase, “наследуются” BDE, и поэтому этими же особенностями обладает и Delphi.

Алиасы

Таблицы сохраняются в базе данных. Некоторые СУБД сохраняют базу данных в виде нескольких отдельных файлов, представляющих собой таблицы (в основном, все локальные СУБД), в то время как другие состоят из одного файла, который содержит в себе все таблицы и индексы (InterBase).

Например, таблицы dBase и Paradox всегда сохраняются в отдельных файлах на диске. Каталог, содержащий dBase *.DBF* файлы или Paradox *.DB* файлы, рассматривается как база данных. Другими словами, любой каталог, содержащий файлы в формате Paradox или dBase, рассматривается Delphi как единая база данных. Для переключения на другую базу данных нужно просто переключиться на другой каталог.

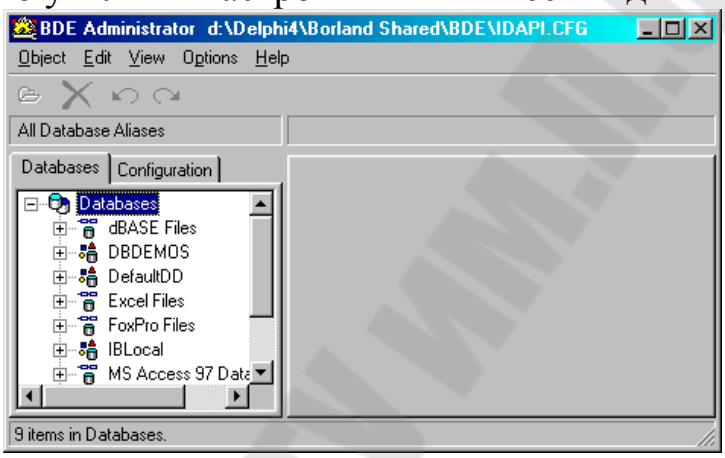
Удобно не просто указывать путь доступа к таблицам базы данных, а использовать для этого некий заменитель - псевдоним, называемый *алиасом*. Он сохраняется в отдельном конфигурационном файле в произвольном месте на диске и позволяет исключить из программы прямое указание пути доступа к базе данных.

Такой подход дает возможность располагать данные в любом месте, не перекомпилируя при этом программу.

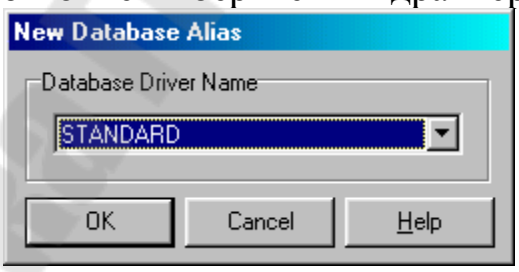
Кроме пути доступа, в алиасе указываются тип базы данных, языковый драйвер и много другой управляющей информации. Поэтому использование алиасов позволяет легко переходить от локальных баз данных к SQL-серверным базам (естественно, при выполнении требований разделения приложения на клиентскую и серверную части).

Для создания алиаса запустите утилиту конфигурации BDE (программу BDEADMIN.EXE), находящуюся в каталоге, в котором располагаются динамические библиотеки BDE.

Главное окно утилиты настройки BDE имеет вид



Для создания алиаса выберите в меню “Object” пункт “New”. В появившемся диалоговом окне выберите имя драйвера базы данных.



Тип алиаса может быть стандартным (STANDARD) для работы с локальными базами в формате dBase или Paradox или соответствовать наименованию SQL-сервера (InterBase, Sybase, Informix, Oracle и т.д.).

После создания нового алиаса следует дать ему имя. Это можно сделать с помощью подпункта “Rename” меню “Object”. Однако просто создать алиас не достаточно. Вам нужно указать дополнительную информацию, содержание которой зависит от типа выбранной базы данных.

Например, для баз данных Paradox и dBase (STANDARD) требуется указать лишь путь доступа к данным, имя драйвера и флаг ENABLE BCD, который определяет, транслирует ли BDE числа в двоично-десятичном формате (значения двоично-десятичного кода устраняют ошибки округления):

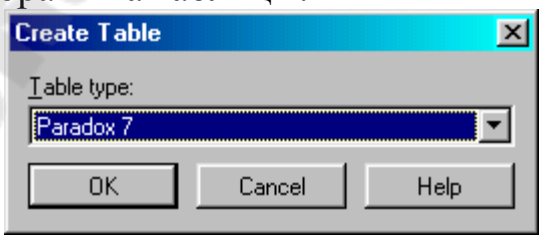
TYPE	STANDARD
DEFAULT DRIVER	PARADOX
ENABLE BCD	FALSE
PATH	c:\users\data

SQL-сервер InterBase и другие типы баз данных требуют задания большого количества параметров, многие из которых можно оставить установленными по умолчанию.

Утилита Database Desktop

Database Desktop - это утилита, во многом похожая на Paradox, которая поставляется вместе с Delphi для интерактивной работы с таблицами различных форматов локальных баз данных - Paradox и dBase, а также SQL-серверных баз данных InterBase, Oracle, Informix, Sybase (с использованием SQL Links). Исполняемый файл утилиты называется DBD32.EXE. Для запуска Database Desktop просто дважды щелкните по ее иконке.

После старта Database Desktop выберите команду меню File|New|Table для создания новой таблицы. Перед Вами появится диалоговое окно выбора типа таблицы.



Вы можете выбрать любой формат из предложенного, включая различные версии одного и того же формата.

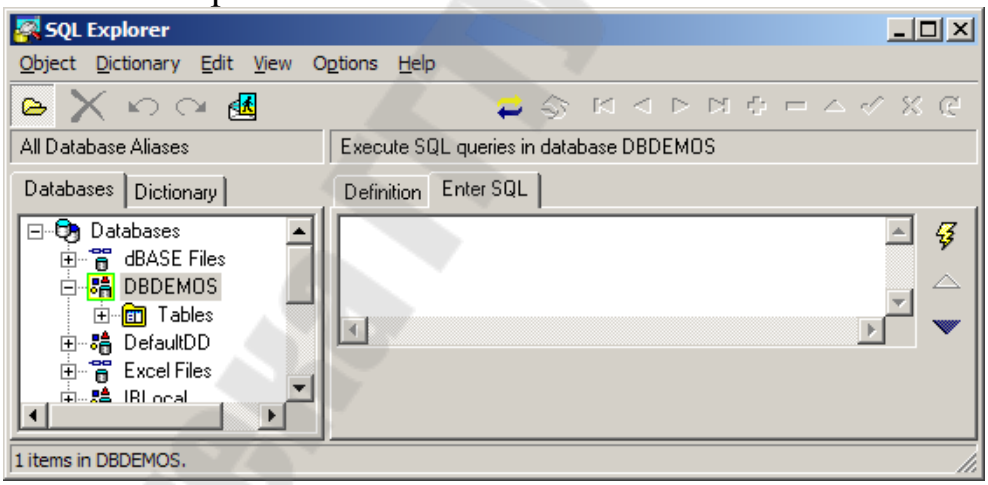
После выбора типа таблицы Database Desktop представит Вам диалоговое окно, специфичное для каждого формата, в котором Вы сможете определить поля таблицы и их тип.

Следующий (после выбора имени поля) шаг состоит в задании типа поля. Типы полей очень сильно различаются друг от друга, в зависимости от формата таблицы. Для получения списка типов полей перейдите к столбцу “Type”, а затем нажмите пробел или щелкните правой кнопкой мышки.

Создание таблиц с помощью SQL

Database Desktop не обладает всеми возможностями по управлению SQL-серверными базами данных. Поэтому с помощью Database Desktop удобно создавать или локальные базы данных или только простейшие SQL-серверные базы данных, состоящие из небольшого числа таблиц, не очень сильно связанных друг с другом.

Если же необходимо создать базу данных, состоящую из большого числа таблиц, имеющих сложные взаимосвязи, можно воспользоваться языком SQL. При этом можно воспользоваться SQL Explorer в Delphi, каждый раз посылая по одному SQL-запросу, а можно записать всю последовательность SQL-предложений в один так называемый скрипт и послать его на выполнение.

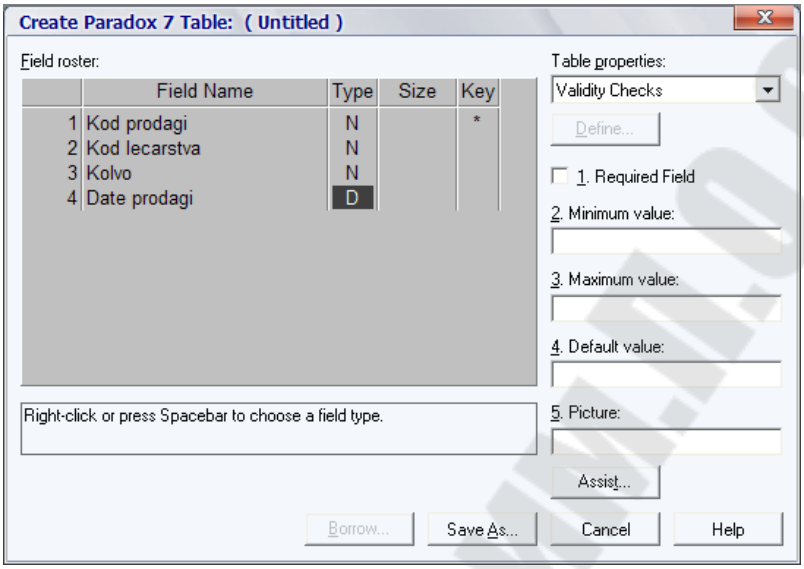


Задание

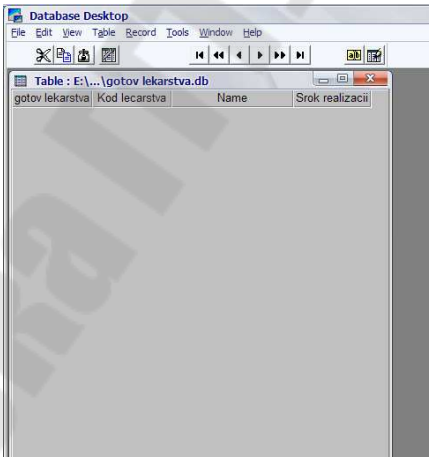
1. Для заданной предметной области разработать структуру (полей) записей БД;
2. Создать файлы базы данных в Database Desktop;
3. Заполнить начальными записями базу данных в Database Desktop. Каждая таблица должна содержать не менее 10 записей.

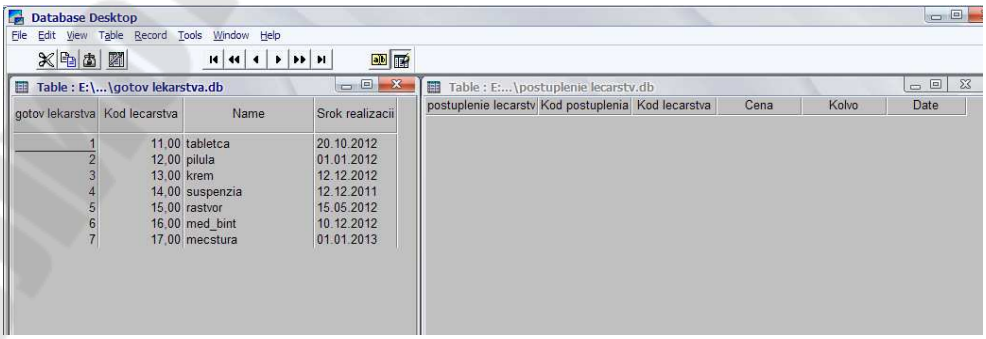
Пояснения к выполнению работы

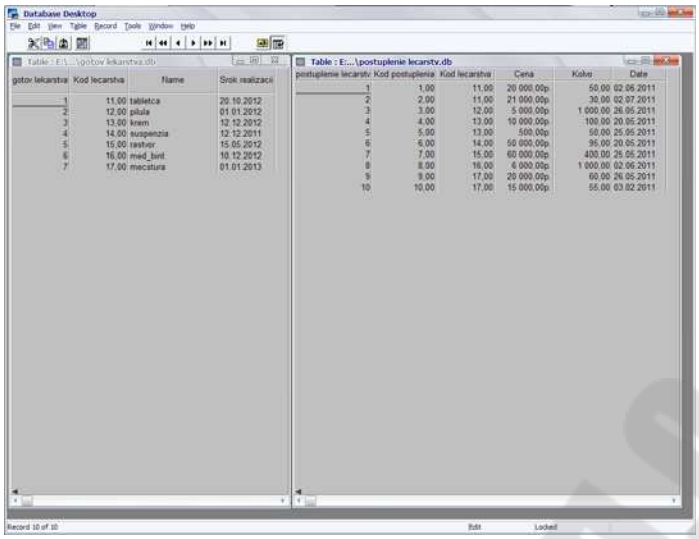
1. Вызвать утилиту Database Desktop, file/new table, создать таблицы с полями

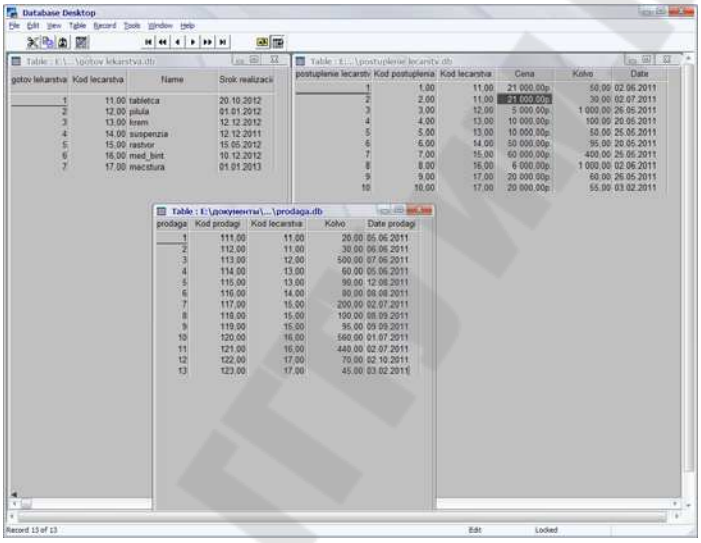


2. Заполнить данными созданные таблицы:
File/open table









3. Создать алиас с помощью BDEAdmin или SQL Explorer

Контрольные вопросы

1. Понятие базы данных. Модели баз данных.
2. Архитектура СУБД.
3. Реляционные базы данных. Первичные ключи и индексы.
4. Избыточность данных.
5. Архитектура баз данных Delphi. Драйверы баз данных.
6. Создание таблиц. Утилита Database Desktop.
7. Работа с утилитой SQL Explorer.
8. Основные свойства, методы и события класса TField.

Лабораторная работа № 3

Тема: Разработка визуального приложения для работы с БД. Компоненты доступа к БД (TDataModule, Table, TDataSource). Визуальные компоненты для работы с БД (TDBGrid, TBDNavigator)

Цель работы: изучит начальные этапы создания визуального приложения для работы с БД. Обеспечить взаимодействие набора данных с компонентами отображения данных.

Теоретические сведения

Чтобы начать использовать эти навигационные методы, Вы должны поместить TTable, TDataSource и TDBGrid на форму. Присоедините DBGrid1 к DataSource1, и DataSource1 к Table1. Затем установите свойства таблицы:

- в DatabaseName имя подкаталога, где находятся демонстрационные таблицы (или псевдоним DBDEMOS);
- в TableName установите имя таблицы CUSTOMER.

Если Вы запустили программу, которая содержит видимый элемент TDBGrid, то увидите, что можно перемещаться по записям таблицы с помощью полос прокрутки (scrollbar) на нижней и правой сторонах DBGrid.

Однако, иногда нужно перемещаться по таблице “программным путем”, без использования возможностей, встроенных в визуальные компоненты.

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

- После того, как Вы открыли файл;
- После того, как Вы вызывали TDataSet.First;
- После того, как вызов TDataSet.Prior не выполняется.

Первые два пункта - очевидны. Когда Вы открываете таблицу, Delphi помещает Вас на первую запись; когда Вы вызываете метод First, Delphi также перемещает Вас в начало таблицы. Третий пункт, однако, требует небольшого пояснения: после того, как Вы вызывали метод Prior много раз, Вы могли добраться до начала таблицы, и следующий вызов Prior будет неудачным - после этого BOF и будет возвращать True.

Следующий код показывает самый общий пример использования Prior, когда Вы попадаете к началу файла:

```
while not Table.Bof do begin
  DoSomething;
  Table1.Prior;
end;
```

Все сказанное относительно BOF также применимо и к EOF. Другими словами, код, приведенный ниже показывает простой способ пробежать по всем записям в a dataset:

```
Table1.First;
while not Table1.EOF do begin
  DoSomething;
  Table1.Next;
end;
```

Классическая ошибка в случаях, подобных этому: Вы входите в цикл while или repeat, но забываете вызывать Table1.Next:

```
Table1.First;
repeat
  DoSomething;
until Table1.EOF;
```

Если Вы случайно написали такой код, то ваша машина зависнет. Также, этот код мог бы вызвать проблемы, если Вы открыли пустую таблицу. Так как здесь используется цикл **repeat**, DoSomething был бы вызван один раз, даже если бы нечего было обрабатывать. Поэтому, лучше использовать цикл **while** вместо **repeat** в ситуациях подобных этой.

EOF возвращает True в следующих трех случаях:

- После того, как Вы открыли пустой файл;
- После того, как Вы вызывали TDataSet.Last;
- После того, как вызов TDataSet.Next не выполняется.

Единственная навигационная процедура, которая еще не упоминалась - MoveBy, которая позволяет Вам переместиться на N записей вперед или назад в таблице. Если Вы хотите переместиться на две записи вперед, то напишите:

```
MoveBy(2);
```

И если Вы хотите переместиться на две записи назад, то:

MoveBy(-2);

При использовании этой функции Вы должны всегда помнить, что DataSet - это изменяющиеся объекты, и запись, которая была пятой по счету в предыдущий момент, теперь может быть четвертой или шестой или вообще может быть удалена...

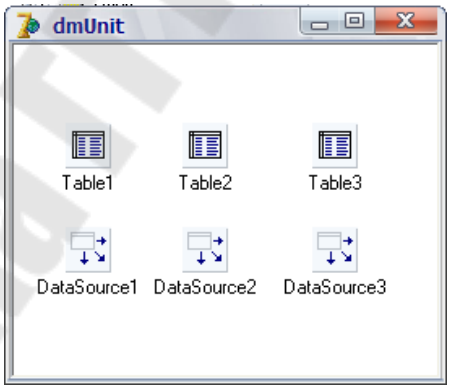
Prior и Next - это простые функции, которые вызывают MoveBy.

Задание

Разработать визуально приложение, обеспечивающее взаимодействие набора данных с компонентами отображения данных. В качестве основного приложения взять наработки лабораторной работы № 1. Базу данных взять из лабораторной работы № 2.

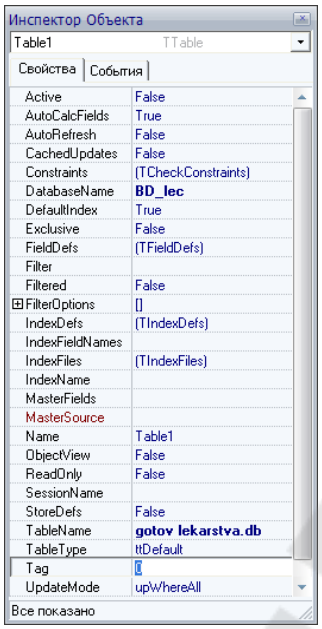
Пояснения к выполнению работы

1. Изучить свойства заданных компонент.
2. Открыть созданный проект в л.р.№1, добавить TDataModule. Разместить на нем необходимое количество компонент Ttable и TDataSource.

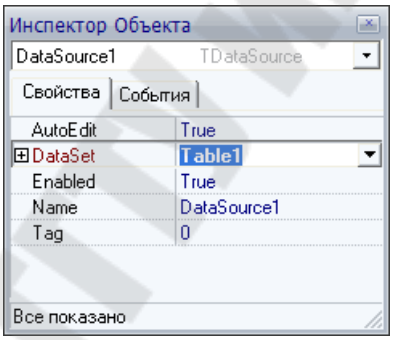


3. Установить свойства компонент, например:

Для Table1:

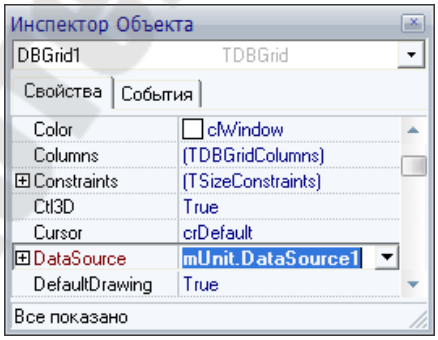


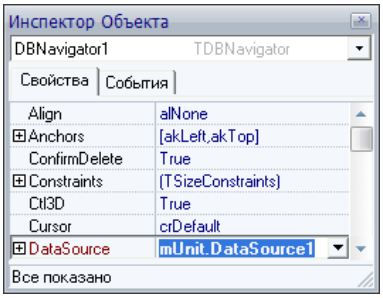
Для DataSource1:



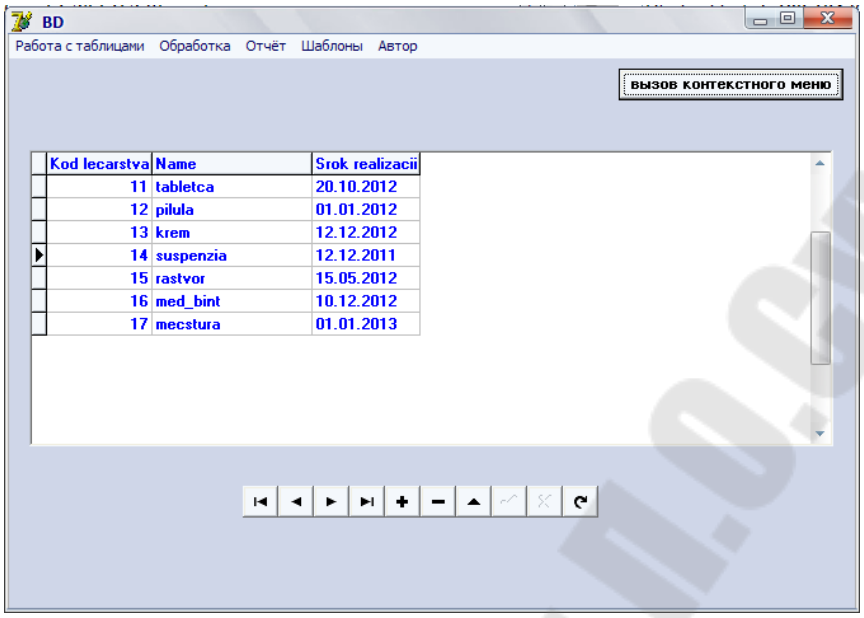
Аналогично для всех остальных.

4. Добавить процедуры обработки событий для просмотра таблиц БД по главному и контекстному меню. Для этого на главной форме разместить компоненты TDBGrid и BDNavigator. Установить свойства компонент:





5. Результат процедуры обработки события для просмотра таблицы БД «Аптека»:



Контрольные вопросы

1. Типы данных поля, виды полей. Объекты для реальных полей
2. Проверка правильности введённого в поле значения.
3. Понятие и состояние набора данных.
4. Навигация по набору данных.
5. Компонент TDBNavigator.
6. Компонент TDBGrid. Создание объектов-столбцов.
7. Пустые столбцы. Формирование списка возможных значений столбца.
8. Управление отображением данных в TDBGrid.
9. Дополнительные возможности компонента TDBGrid.

Лабораторная работа № 4

Тема: Работа с данными (редактирование, добавление, удаление). Организация фильтрации, сортировки и поиска записей. Создание вычисляемых полей

Цель работы: Разработать визуальное приложение, позволяющее добавлять, редактировать и удалять записи из БД. Научиться формировать вычисляемые поля с помощью класса TField,

фильтры для отбора данных, сортировать данные по значениям любого заданного столбца с использованием основного и дополнительных индексов, организовывать поиск по значениям любого заданного столбца.

Теоретические сведения

Манипулирование данными

Ниже приведена последовательность действий, которые необходимо выполнить для редактирования одного или нескольких полей в текущей записи.

1. Вызовите метод Edit() набора данных для переключения в режим редактирования (Edit).

2. Назначьте новые значения требуемым полям.

3. Внесите изменения в набор данных посредством вызова метода Post() или перемещения на новую запись, в результате чего изменения будут внесены автоматически.

Например, типичная процедура изменения записи выглядит следующим образом:

```
TableI.Edit;
Table1[ 'Age'] := 23;
Tablel.Post;
```

Как и в случае редактирования, вставлять или добавлять записи в конец набора данных можно следующим образом.

1. Вызовите метод Insert () или Append () набора данных для переключения в режим вставки (Insert) или добавления (Append).

2. Присвойте значения полям новой записи набора данных.

3. Внесите новую запись в набор данных посредством вызова метода Post() или перемещения на новую запись, в результате чего изменения будут внесены автоматически.

Если потребуется отменить внесенные в запись, но еще не зафиксированные в наборе данных изменения, это можно сделать посредством вызова метода Cancel (). Например, следующий фрагмент кода отменяет выполненное редактирование до внесения изменений в таблицу:

```
TableI.Edit;
Table1[ 'Age'] := 23;
Table1.Cancel;
```

Метод Cancel () отменяет внесенные, но не зафиксированные изменения и возвращает набор данных из режима Edit, Append или Insert в режим Browse.

Метод Delete() удаляет текущую запись из набора данных. Например, в следующем фрагменте кода удаляется последняя запись в таблице:

Tablel.Last; Table1.Delete;

Ограничения вводимых значений

Несколько возможностей ограничения предоставляют свойства полей, которые можно увидеть, сделав двойной щелчок на компоненте Table и выделив в окне Редактора Полей требуемое поле.

Для числовых полей имеются свойства. Value и MaxValue, устанавливающие допустимые пределы вводимых в поле значений. При нарушении этих пределов будет генерироваться исключение EDatabaseError, которое лучше перехватывать в приложении, чтобы выдать пользователю понятное пояснение на русском языке.

Другая возможность ограничения — использование свойств CustomConstraint и ConstraintErrorMessage.

Свойство CustomConstraint позволяет написать ограничение на значение поля в виде строки SQL

Если вы задали свойство CustomConstraint, то необходимо задать и своё. ConstraintErrorMessage. Оно содержит строку текста, который будет показано пользователю в случае, если он вводит данные, не удовлетворяющие поставлен ограничениям.

Еще одна возможность проверять данные на уровне поля — использовать обработку события поля OnValidate. Это событие возникает перед записью введенного значения поля в буфер текущей записи. Тут можно предусмотреть любые проверки, при появлении недопустимых данных выдать пользователю сообщение и пример, сгенерировать исключение EAbort функцией Abort.

Если все нормально, то после события OnValidate возникает еще событие OnChange, в обработчике которого тоже еще не поздно сгенерировать исключение.

Описанные выше способы проверки данных относятся к конкретному полю. Имеется также возможность осуществлять проверку на уровне записи, анализирую; различные ее поля.

Для этого служит свойство Constraints компонента Table, нажмите кнопку в этом свойстве, и перед вами откроется окно.

Щелкая в нем на кнопке Add, вы можете занести в свойство Constraints набор ограничений. Каждое из них представляет собой самостоятельный объект.

Выделив одно из ограничений, вы увидите в Инспекторе Объектов его свойства, свойство CustomConstraint представляет собой строку SQL, определяющую допустимые значения. Свойство ErrorMessage определяет строку текста, которая будет объявлена пользователю в случае нарушения ограничений.

Фильтрация набора данных

Один из наиболее общих механизмов фильтрации, используемых в Delphi, — ограничение представления набора данных до некоторых указанных записей.

Этот процесс выполняется следующим образом.

1. Назначьте процедуру событию OnFilterRecord набора данных. В эту процедуру следует поместить операторы, выполняющие отбор записей на основе значений одного или нескольких полей.

2. Установите свойство Filtered набора данных равным True.

На первом этапе создадим обработчик события OnFilterRecord для этой таблицы. В данном случае будут отбираться только те записи, в которых значение поле Company начинается с прописной буквы S.

Текст этой процедуры выглядит следующим образом:

```
procedure TForml.TablelFilterRecord(DataSet: TDataSet; var Accept: Boolean);
var
    FieldVal: String;
begin
    FieldVal:= DataSet['Company']; // Получение значения поля
                // Company
    Accept:= FieldVal[l] = 'S'; // Принять запись,
            // если поле начинается с буквы S
end;
```

Организация поиска

Поиск данных производиться по одной из команд меню Поиск:

- по фамилии (J=5);
- быстрый по фамилии (J=6);
- ближайшей подходящей фамилии (J=7);
- постепенный поиск фамилии;

- по фамилии и стипендии (J=8).

Значение J определяется из свойства Tag соответствующих пунктов меню. Поэтому значения свойства Tag должны быть такие же, как записанные выше значения J для соответствующих пунктов меню.

Для первых 4-х поисков и последнего используется один метод, который создается для первого поиска и имеет следующий код:

```
with Sender as TmenuItem do begin
    J:=Tag;
    Panel1.Visible:=True;
   Edit1.SetFocus;
   if J=8 then begin Panel2.Visible:=True;
    MaskEdit1.SetFocus;
  Label7.Caption:='Введите размер стипендии';
end;
```

В приведенном выше методе становится видимой панели Panel1 и Panel2 для последнего поиска, определяется значение J.

Дальнейший поиск производиться действиями, реализованными в методе нажатия кнопки Искать и который имеет следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
procedure S1;
begin
MessageDLG('Запись найдена',mtInformation,[mbOK],0);
end;
procedure S2;
begin BEEP;
MessageDLG('Запись не найдена',mtInformation,[mbOK],0);
end;
begin
```

```
Case J OF
                    5: BEGIN TABLE1.IndexFieldNames:='FIO';
                    Table1.SetKey;
                    Table1.FieldByName('FIO').AsString:=Edit1.Text;
                     if Table1.GotoKey then s1 else s2;
                      end;
                    6:                                    begin                                  if
Table1.Locate('FIO',Edit1.Text,[LoCaseInsensitive,LoPartialKey])
                          then s1 else s2;  end;
                    7: begin
                          Table1.IndexFieldNames:='FIO';
                          Table1.SetKey;
                          Table1.FieldByName('FIO').AsString:=Edit1.Text;
                          Table1.GotoNearest;
                      end;
                     8: begin
                          if
Table1.Locate('FIO;RS',VarArrayof([Edit1.text,MaskEdit1.Text]),
                                [LoCaseInsensitive,LoPartialKey]) then s1 else s2;
                        Panel2.Visible:=False;
                       end;
              end;
                    Panel1.visible:=False;
      end;
```

LoCaseInsensitive – регистр букв не учитывается

LoPartialKey – допускается частичное совпадение значений

Для постепенного поиска работает метод, созданный для этого пункта и имеющий код:

```
Panel3.Visible:=True;
Edit2.SetFocus;
```

и метод

```
procedure TForm1.Edit2Change(Sender: TObject);
begin
TABLE1.IndexFieldNames:='FIO';
Table1.FindNearest([Edit2.Text]);
end;
```

По двойному щелчку в Edit2 вызывается метод

```
procedure TForm1.Edit2DblClick(Sender: TObject);
begin
Panel3.Visible:=False;
end;
```

Создание вычисляемых полей

Например, допустим, что в набор данных необходимо добавить поле, отображающее для каждой строки в таблице Postuplenie lecarstv цену лекарства, составляющую 50% от исходной.

Выберите в контекстном меню окна редактора полей команду New Field. На экране раскроется диалоговое окно New Field. В поле Name этого окна введите имя нового поля— TCena.

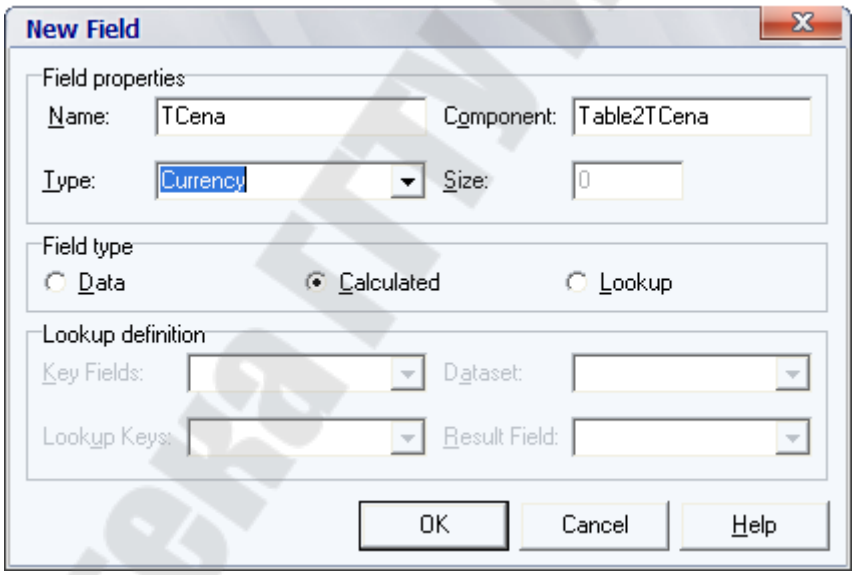
Тип этого поля— Currency, поэтому в раскрывающемся списке Type выберите именно это значение.

В группе Field Type установите переключатель в положение Calculated и щелкните на кнопке OK. Новое поле появится в сетке, но пока не будет содержать никаких данных.

Чтобы заполнить новое поле данными, необходимо назначить требуемый метод событию OnCalcFields объекта Tablel. В тексте обработчика этого события полю TCTce следует просто присвоить значение, равное 50% от существующего значения поля Cena.

Соответствующий текст метода обработки события Tablel.OnCalcFields показан ниже:

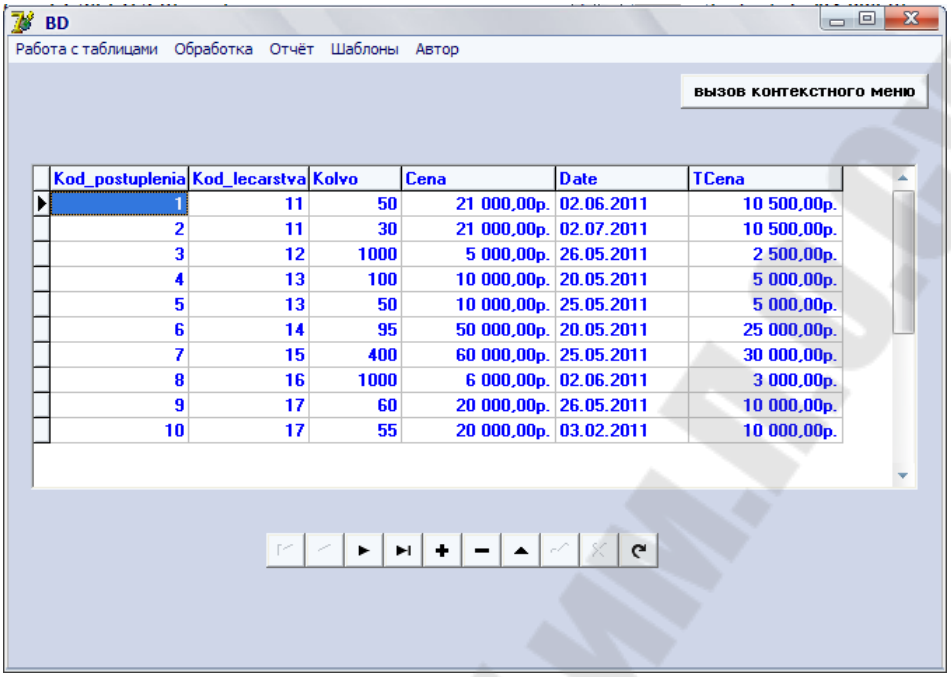
```
procedure TdmUnit.TCena(DataSet: TDataSet); //формирование вычисляемого поля
begin
  DataSet['TCena'] := DataSet['Cena'] * 0.5;
end;
```



Задание

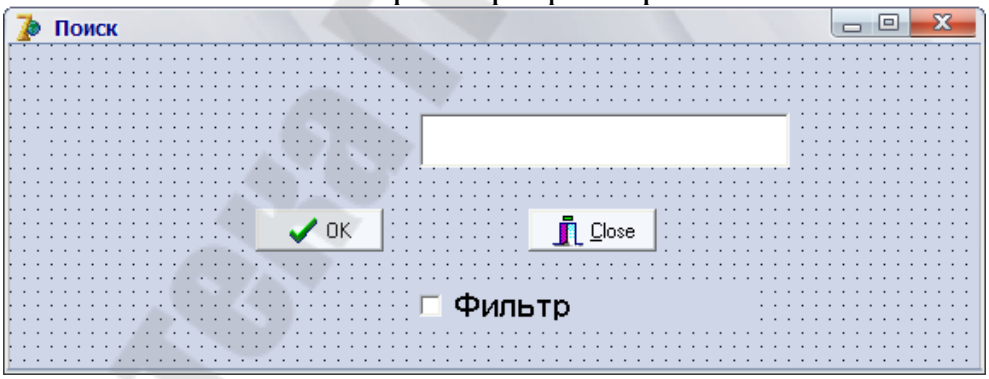
Добавить в приложение, разработанное в лабораторной работе № 3, функционал, позволяющий добавлять, редактировать и удалять записи из БД. Сформировать вычисляемые поля с помощью класса TField, фильтры для отбора данных. Приложение также должно иметь возможность сортировать данные по значениям любого заданного столбца с использованием основного и дополнительных индексов, организовывать поиск по значениям любого заданного столбца.

Пояснения к выполнению работы

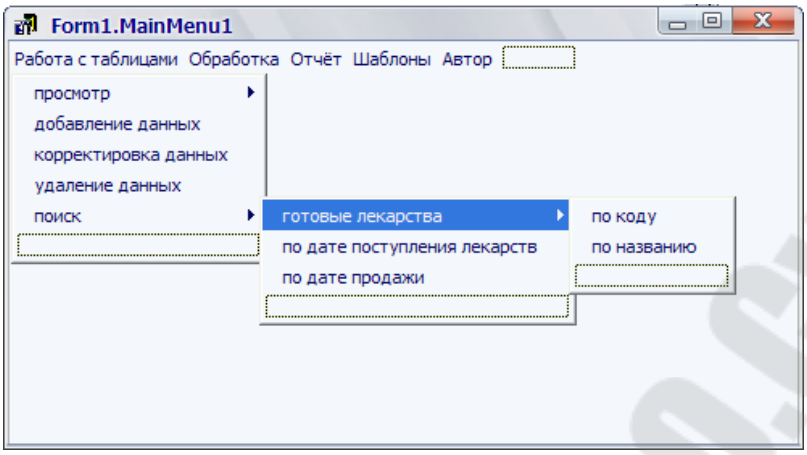


Kod_postuplenia	Kod_lecarstva	Kolvo	Cena	Date	TCena
1	11	50	21 000,00р.	02.06.2011	10 500,00р.
2	11	30	21 000,00р.	02.07.2011	10 500,00р.
3	12	1000	5 000,00р.	26.05.2011	2 500,00р.
4	13	100	10 000,00р.	20.05.2011	5 000,00р.
5	13	50	10 000,00р.	25.05.2011	5 000,00р.
6	14	95	50 000,00р.	20.05.2011	25 000,00р.
7	15	400	60 000,00р.	25.05.2011	30 000,00р.
8	16	1000	6 000,00р.	02.06.2011	3 000,00р.
9	17	60	20 000,00р.	26.05.2011	10 000,00р.
10	17	55	20 000,00р.	03.02.2011	10 000,00р.

1. Добавить в проект, разработанный в л.р. №3 формы для организации фильтра. На форме разместить следующие компоненты: 2 TBitBtn для начала фильтра и закрытия формы, TLable и TEdit для ввода названия и значения параметра фильтра.



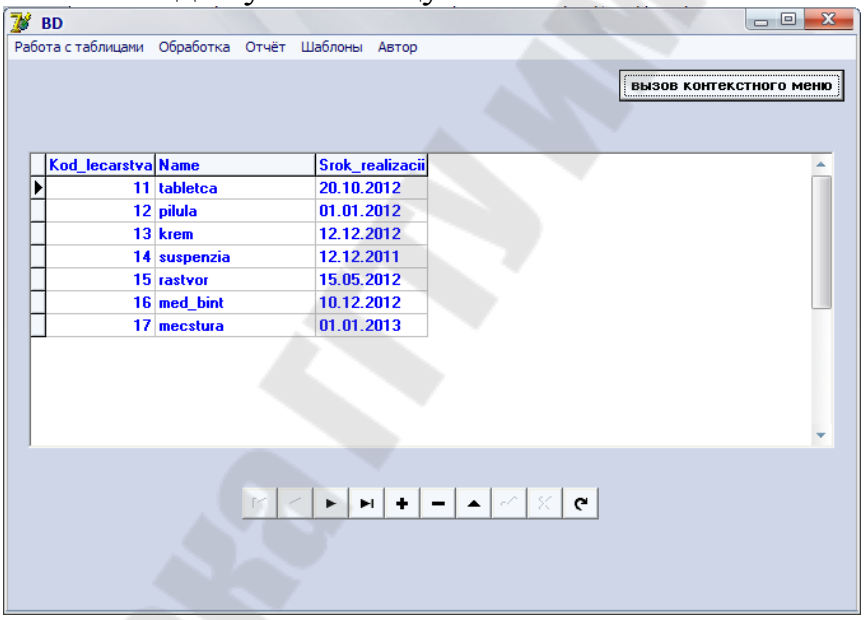
2. Добавить в главное меню подразделы в раздел «Поиск» Например:



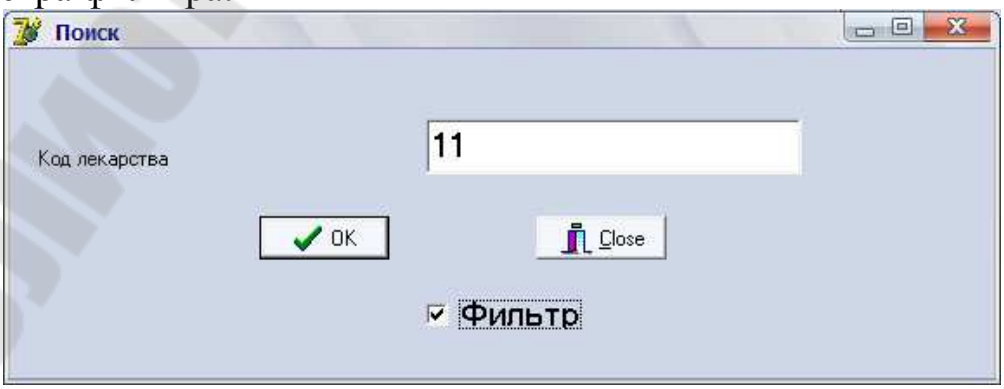
Для каждого подраздела написать соответствующие процедуры обработки событий

Пример результата поиска:

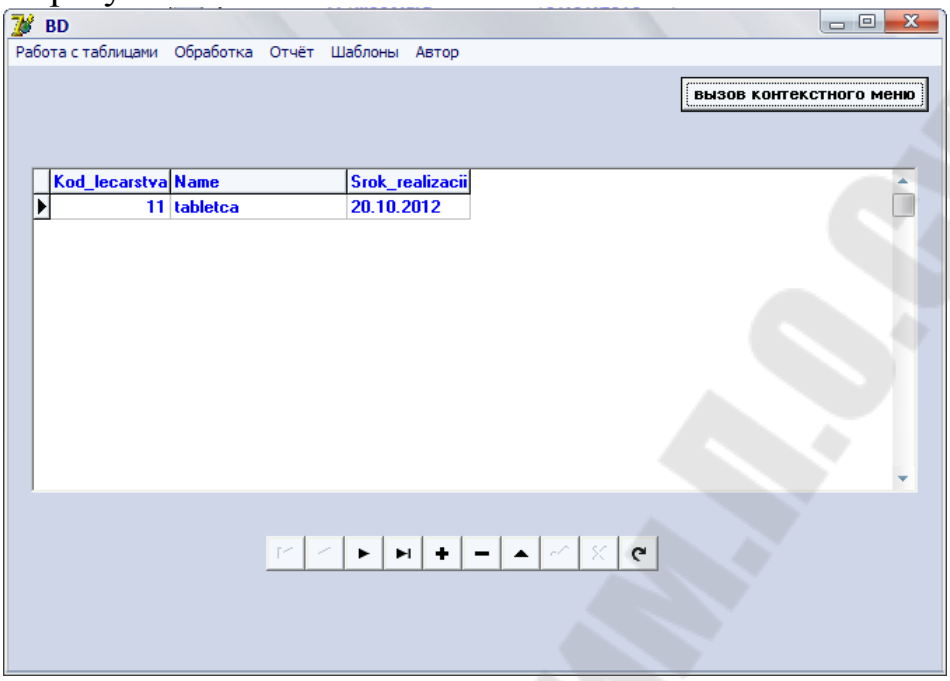
1) Выбираем необходимую таблицу



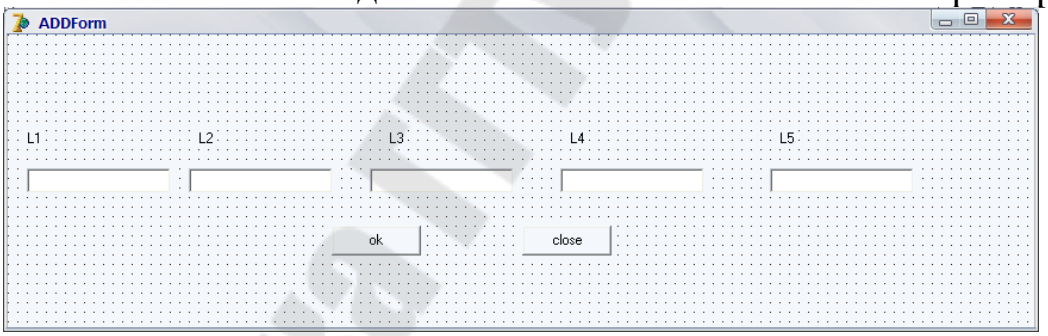
Затем в меню выбираем необходимый фильтр и вводим значение параметра фильтра:



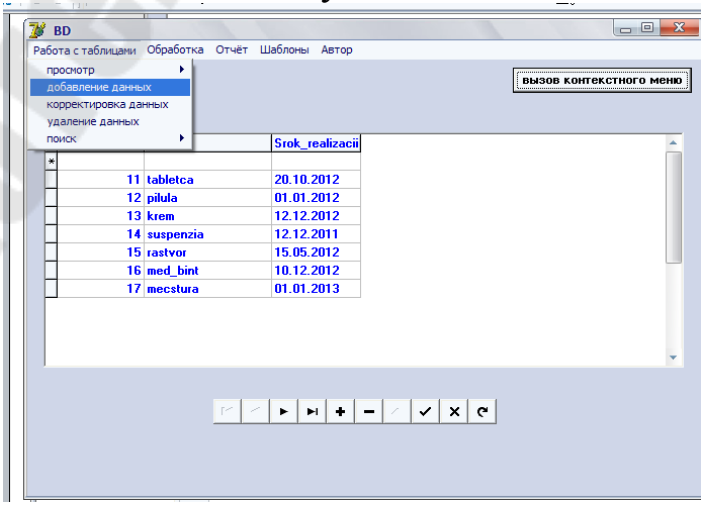
Получаем результат:



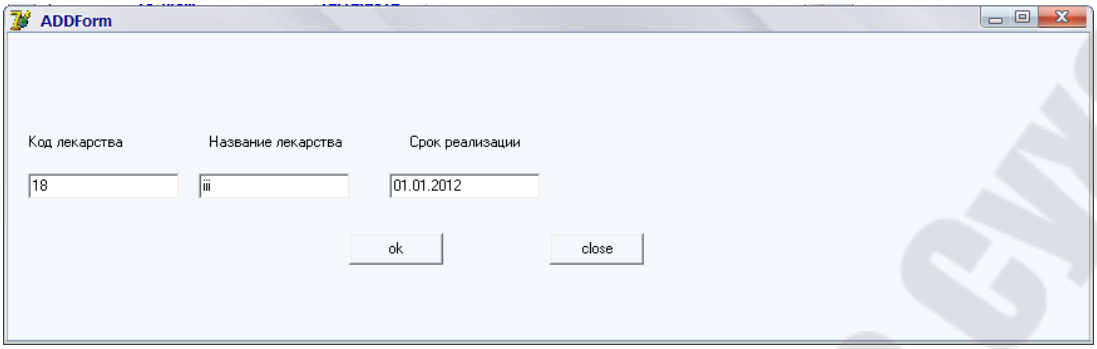
2) Добавить форму для добавления и корректировки данных БД. Разместить на ней необходимое количество Tlabel и Tedit. Пример:



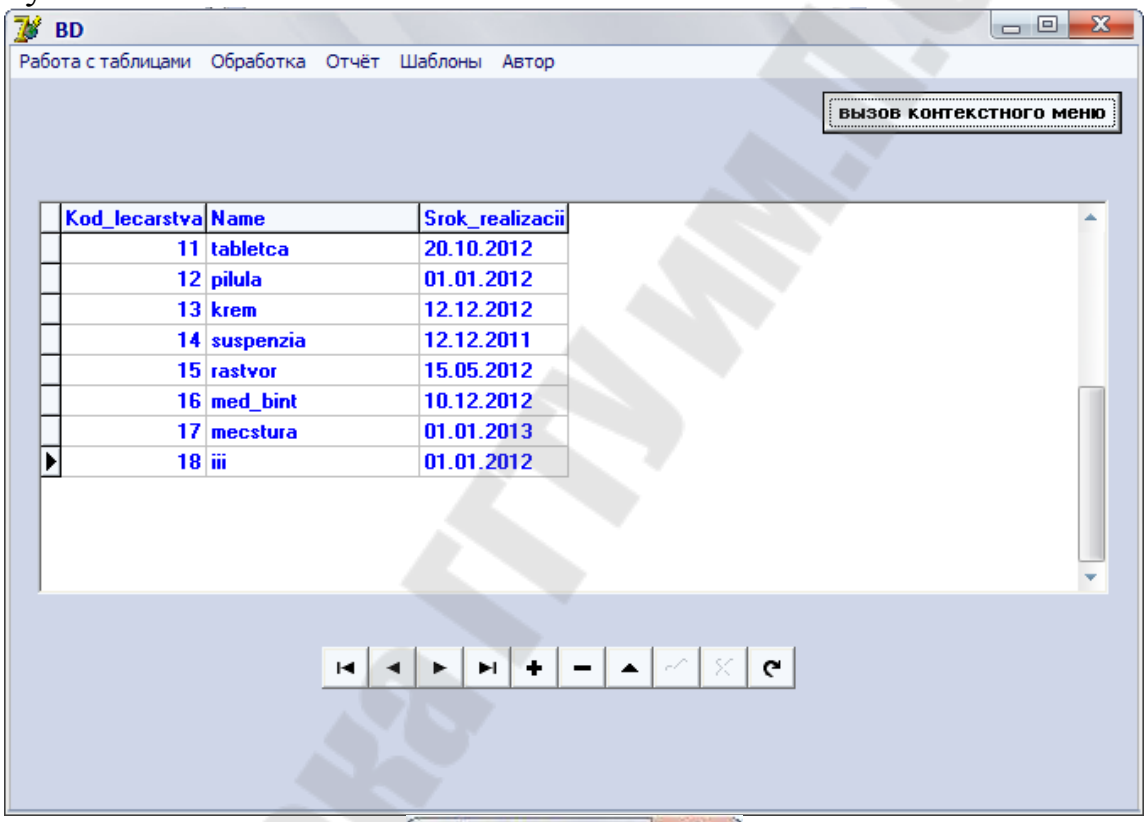
Пример результата добавления и удаления записи:

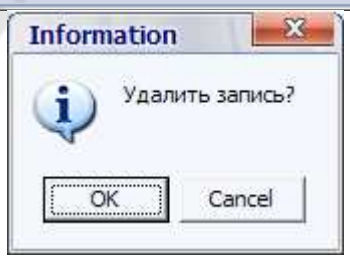


Вводим новые данные:



Результат:





Контрольные вопросы

1. Обращение к значению поля. Создание вычисляемых полей.
2. Навигация по набору данных.
3. Внесение изменений в набор данных.

4. Работа с индексами.
5. Фильтрация записей в наборе данных.
6. Поиск записей в наборе данных.

Лабораторная работа № 5

Тема: Создание LookUp-полей. Работа со связанными (MasterSource) таблицами

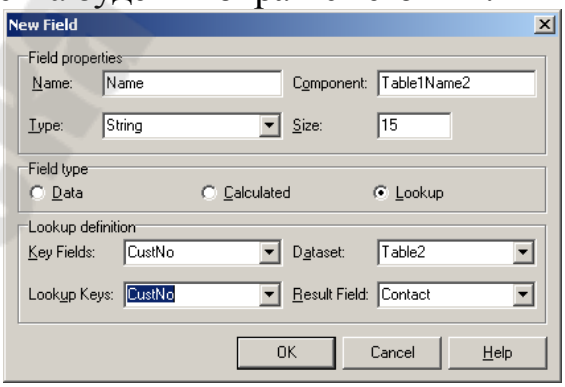
Цель работы: ознакомиться со способами создания Look-Up-полей. Научится организовывать визуальные приложения, отображающие данные из связанных таблиц.

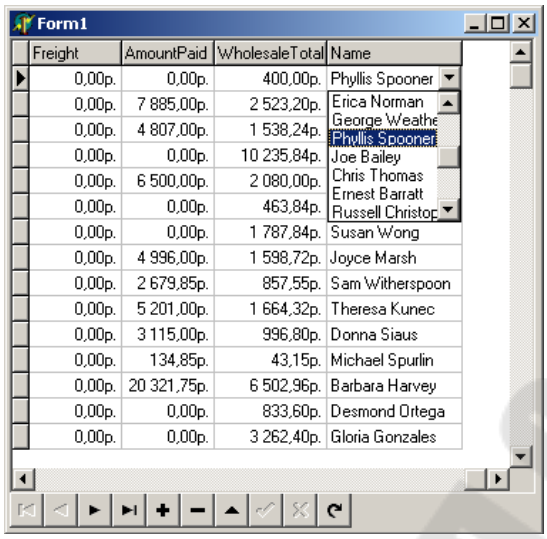
Теоретические сведения

Подстановочные поля

Подстановочные (lookup) поля позволяют создавать в наборе данных такие поля, значения которых будут выбираться из другого набора данных.

Для иллюстрации сказанного добавим такое поле к текущему проекту. Вряд ли по номеру клиента (поле CustNo таблицы ORDERS) можно будет вспомнить его имя. Поэтому целесообразно добавить к таблице Tablel подстановочное поле, связанное с таблицей CUSTOMER, из которой по номеру клиента будет выбираться его имя.





Вначале поместите в форму второй объект TTable и присвойте его свойству DatabaseName значение DBDEMOS, а свойству TableName значение CUSTOMER.DB. Это будет объект Table2.

Затем вновь откройте окно New Field, для чего выберите команду New Field в контекстном меню окна редактора поля. Присвойте новому полю имя CustName и тип String.

Размер поля установите равным 15 символам. Не забудьте установить переключатель в группе Field Type в положение Lookup.

В списке Dataset этого диалогового окна выберите значение Table2 — именно этот набор данных необходимо просматривать.

В обоих списках Key Fields и Lookup Keys этого диалогового окна выберите значение CustNo — это то общее поле, по значению которого будет выполняться поиск.

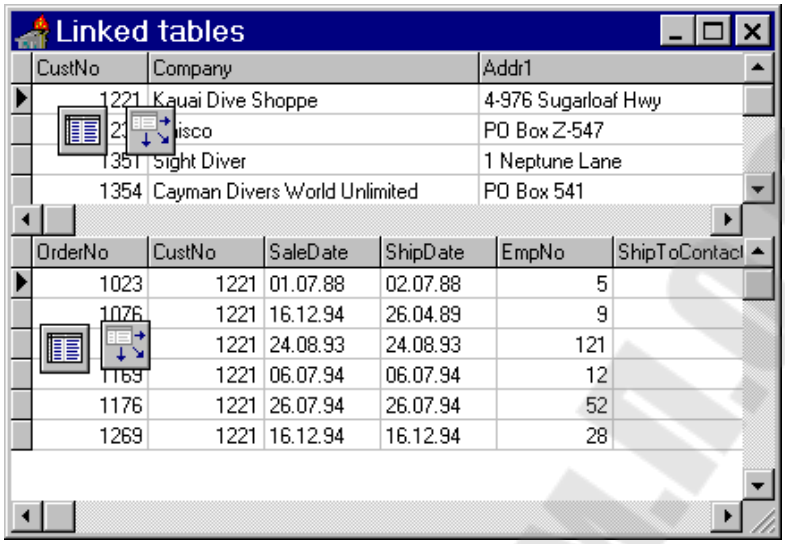
И, наконец, в списке Result выберите значение Contact — именно это поле необходимо отображать в нашем наборе данных.

Создание Связанных курсоров (Linked cursors)

Связанные курсоры позволяют программистам определить отношение один ко многим (one-to-many relationship). Например, иногда полезно связать таблицы CUSTOMER и ORDERS так, чтобы каждый раз, когда пользователь выбирает имя заказчика, то он видит список заказов связанных с этим заказчиком. Иначе говоря, когда пользователь выбирает запись о заказчике, то он может просматривать только заказы, сделанные этим заказчиком.

Чтобы создать программу заново, поместите два TTable, два TDataSources и два TDBGrid на форму. Присоедините первый набор

таблице CUSTOMER, а второй к таблице ORDERS. Программа в этой стадии имеет вид



Следующий шаг должен связать таблицу ORDERS с таблицей CUSTOMER так, чтобы Вы видели только те заказы, которые связанные с текущей записью в таблице заказчиков. В первой таблице заказчик однозначно идентифицируется своим номером - поле CustNo. Во второй таблице принадлежность заказа определяется также номером заказчика в поле CustNo. Следовательно, таблицы нужно связывать по полю CustNo в обоих таблицах (поля могут иметь различное название, но должны быть совместимы по типу).

Для этого, Вы должны сделать три шага, каждый из которых требует некоторого пояснения:

1. Установить свойство Table2.MasterSource = DataSource1
2. Установить свойство Table2.MasterField = CustNo
3. Установить свойство Table2.IndexName = CustNo

Если Вы теперь запустите программу, то увидите, что обе таблицы связаны вместе, и всякий раз, когда Вы перемещаетесь на новую запись в таблице CUSTOMER, Вы будете видеть только те записи в таблице ORDERS, которые принадлежат этому заказчику.

Свойство MasterSource в Table2 определяет DataSource, от которого Table2 может получить информацию. То есть, оно позволяет таблице ORDERS знать, какая запись в настоящее время является текущей в таблице CUSTOMERS.

Но тогда возникает вопрос: Какая еще информация нужна Table2 для того, чтобы должным образом отфильтровать содержимое таблицы ORDERS? Ответ состоит из двух частей:

1. Требуется имя поля, по которому связанны две таблицы.

2. Требуется индекс по этому полю в таблице ORDERS (в таблице ‘многих записей’), которая будет связываться с таблицей CUSTOMER (таблице в которой выбирается ‘одна запись’).

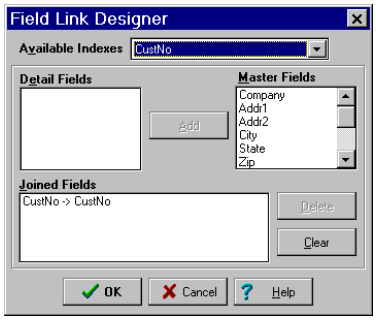
Чтобы правильно воспользоваться информацией описанной здесь, Вы должны сначала проверить, что таблица ORDERS имеет нужные индексы. Если этот индекс первичный, тогда не нужно дополнительно указывать его в поле IndexName, и поэтому Вы можете оставить это поле незаполненным в таблице TTable2 (ORDERS). Однако, если таблица связана с другой через вторичный индекс, то Вы должны явно определять этот индекс в поле IndexName связанной таблицы.

В примере, показанном здесь таблица ORDERS не имеет первичного индекса по полю CustNo, так что Вы должны явно задать в свойстве IndexName индекс CustNo.

Недостаточно, однако, просто упомянуть имя индекса, который Вы хотите использовать. Некоторые индексы могут содержать несколько полей, так что Вы должны явно задать имя поля, по которому Вы хотите связать две таблицы. Вы должны ввести имя ‘CustNo’ в свойство Table2.MasterFields.

Если Вы хотите связать две таблицы больше чем по одному полю, Вы должны внести в список все поля, помещая символ ‘|’ между каждым: Table1.MasterFields := ‘CustNo | SaleData | ShipDate’;

В данном конкретном случае, выражение, показанное здесь, не имеет смысла, так как хотя поля SaleData и ShipDate также индексированы, но не дублируются в таблице CUSTOMER. Поэтому Вы должны ввести только поле CustNo в свойстве MasterFields. Вы можете определить это непосредственно в редакторе свойств, или написать код подобно показанному выше. Кроме того, поле (или поля) связи можно получить, вызвав редактор связей - в Инспекторе Объектов дважды щелкните на свойство MasterFields.



Задание

Усовершенствовать проект, разработанный в лабораторной работе № 4:

1. Создать несколько Look-Up-полей, заменяющих числовые поля соответствующими текстовыми.
2. В соответствии со своим вариантом организовать работу минимум двух связанных таблиц.

Контрольные вопросы

1. Проверка правильности введённого в поле значения.
2. Формирование текстового представления поля.
3. Организация связи таблиц.
4. Понятие ключевых полей.

Лабораторная работа № 6

Тема: Работа с компонентой TQuery

Цель работы: ознакомиться с основными возможностями компоненты TQuery. С помощью компоненты TQuery научиться создавать простые и параметрические запросы на выборку к БД, формировать запросы на вставку, изменение и удаление записей.

Теоретические сведения

Основные понятия о TQuery

При использовании TTable, возможен доступ ко всему набору записей из одной таблицы. В отличие от TTable, TQuery позволяет произвольным образом (в рамках SQL) выбрать набор данных для работы с ним. Во многом, методика работы с объектом TQuery похожа на методику работы с TTable, однако есть свои особенности.

Вы может создать SQL запрос, используя компонент TQuery следующим способом:

1. Назначите Псевдоним (Alias) DatabaseName.

2. Используйте свойство SQL чтобы ввести SQL запрос типа
   “Select * from Country”.
3. Установите свойство Active в True

Если обращение идет к локальным данным, то вместо псевдонима можно указать полный путь к каталогу, где находятся таблицы.

Свойства SQL

Свойство SQL - вероятно, самая важная часть TQuery. Доступ к этому свойству происходит либо через Инспектор Объектов во время конструирования проекта (design time), или программно во время выполнения программы (run time).

Интересней, конечно, получить доступ к свойству SQL во время выполнения, чтобы динамически изменять запрос.

Например, если требуется выполнить три SQL запроса, то не надо размещать три компонента TQuery на форме. Вместо этого можно разместить один и просто изменять свойство SQL три раза.

Наиболее эффективный, простой и мощный способ - сделать это через параметризованные запросы.

Свойство SQL имеет тип TStrings, который означает что это ряд строк, сохраняемых в списке. Список действует также, как и массив, но, фактически, это специальный класс с собственными уникальными возможностями.

При программном использовании TQuery, рекомендуется сначала закрыть текущий запрос и очистить список строк в свойстве SQL:

```
Query1.Close;
Query1.SQL.Clear;
```

Обратите внимание, что всегда можно “безопасно” вызвать Close. Даже в том случае, если запрос уже закрыт, исключительная ситуация генерироваться не будет.

Следующий шаг - добавление новых строк в запрос:

```
Query1.SQL.Add(‘Select * from Country’);
Query1.SQL.Add(‘where Name = ’’Argentina’’’);
```

Метод Add используется для добавления одной или нескольких строк к запросу SQL. Общий объем ограничен только количеством памяти на вашей машине.

Чтобы Delphi отработал запрос и возвратил курсор, содержащий результат в виде таблицы, можно вызвать метод: Query1.Open;

TQuery и Параметры

Delphi позволяет составить “гибкую” форму запроса, называемую параметризованным запросом. Такие запросы позволяют подставить значение переменной вместо отдельных слов в выражениях “where” или “insert”. Эта переменная может быть изменена практически в любое время.

Перед тем, как начать использовать параметризованные запросы, рассмотрим снова одно из простых вышеупомянутых предложений SQL:

```
Select * from Country where Name like ’C%’
```

Можно превратить это утверждение в параметризованный запрос заменив правую часть переменной NameStr:

```
select * from Country where Name like :NameStr
```

В этом предложении SQL, NameStr не является предопределенной константой и может изменяться либо во время дизайна, либо во время выполнения. SQL parser (программа, которая разбирает текст запроса) понимает, что он имеет дело с параметром, а не константой потому, что параметру предшествует двоеточие ":NameStr". Это двоеточие сообщает Delphi о необходимости заменить переменную NameStr некоторой величиной, которая будет известна позже.

Обратите внимание, слово NameStr было выбрано абсолютно случайно. Использовать можно любое допустимое имя переменной, точно также, как выбирается идентификатор переменной в программе.

Есть два пути присвоить значение переменной в параметризованном запросе SQL.

Один способ состоит в том, чтобы использовать свойство Params объекта TQuery.

Второй - использовать свойство DataSource для получения информации из другого DataSet. Вот ключевые свойства для достижения этих целей:

```
property Params[Index: Word];
function ParamByName(const Value: string);
property DataSource;
```

Если подставлять значение параметра в параметризованный запрос через свойство Params, то обычно нужно сделать четыре шага:

1. Закрыть TQuery
2. *Подготовить объект TQuery, вызвав метод Prepare*
3. Присвоить необходимые значения свойству Params
4. Открыть TQuery

Второй шаг выполняется в том случае, если данный текст запроса выполняется впервые, в дальнейшем его можно опустить.

Вот фрагмент кода, показывающий как это может быть выполнено практически:

```
Query1.Close;
Query1.Prepare;
Query1.Params[0].AsString := ‘Argentina’;
Query1.Open;
```

Этот код может показаться немного таинственным. Чтобы понять его, требуется внимательный построчный анализ. Проще всего начать с третьей строки, так как свойство Params является “сердцем” этого процесса.

Params - это индексированное свойство, которое имеет синтаксис как у свойства Fields для TDataSet. Например, можно получить доступ к первой переменной в SQL запросе, адресуя нулевой элемент в массиве Params:

```
Params[0].AsString := ‘”Argentina”’;
```

Если параметризованный SQL запрос выглядит так:

```
select * from Country where Name = :NameStr
```

то конечный результат (т.е. то, что выполнится на самом деле) - это следующее предложение SQL:

```
select * from Country where Name = “Argentina”
```

Все, что произошло, это переменной :NameStr было присвоено значение "Аргентина" через свойство Params. Таким образом, Вы закончили построение простого утверждения SQL.

Если в запросе содержится более одного параметра, то обратиться к ним можно, изменяя индекс у свойства Params

```
Params[1].AsString := ‘SomeValue’;
```

либо используя доступ по имени параметра

```
ParamByName(‘NameStr’).AsString:=’”Argentina”’;
```

Итак, параметризованные SQL запросы используют переменные, которые всегда начинаются с двоеточия, определяя места, куда будут переданы значения параметров.

Прежде, чем использовать переменную Params, сначала можно вызвать Prepare. Этот вызов заставляет Delphi разобрать ваш SQL запрос и подготовить свойство Params так, чтобы оно "было готово принять" соответствующее количество переменных. Можно присвоить значение переменной Params без предварительного вызова Prepare, но это будет работать несколько медленнее.

После того, как Вы вызывали Prepare, и после того, как присвоили необходимые значения переменной Params, Вы должны вызвать Open, чтобы закончить привязку переменных и получить желаемый DataSet. В нашем случае, DataSet должен включать записи, где в поле "Name" стоит "Argentina".

Задание

В соответствии со своим вариантом разработать приложение, позволяющее строить простые и параметрические запросы на выборку к БД, формировать запросы на вставку, изменение и удаление записей.

Контрольные вопросы

1. Основные свойства и методы компонента TQuery.
2. Свойство SQL компонента TQuery. Параметрические запросы.
3. Формирование структуры запроса с помощью SQL Builder (построителя запросов).
4. Оператор SELECT.
5. Агрегатные функции и группировка записей.
6. Вставка, изменение и удаление записей.

Лабораторная работа № 7

Тема: Использование технологии ADO в Delphi

Цель работы: ознакомиться с возможностями разработки визуальных приложений, работающих с БД по технологии ADO.

Теоретические сведения

Для работы с базами в Delphi есть несколько наборов компонентов. Каждый набор очень хорошо подходит для решения определенного круга задач. Все они используют разные технологии доступа к данным и отличаются по возможностям.

ADO [Active Data Objects]

Технология доступа к данным, разработанная корпорацией Microsoft. Рекомендуется использовать для работы с базами данных Microsoft, а именно MS Access или MS SQL Server.

ADOConnection - компонент ADOConnection обеспечивает соединение с базой данных:

ADOTable - компонент ADOTable представляет собой таблицу [все столбцы] базы данных [обеспечивает доступ к таблице] ADODataSet

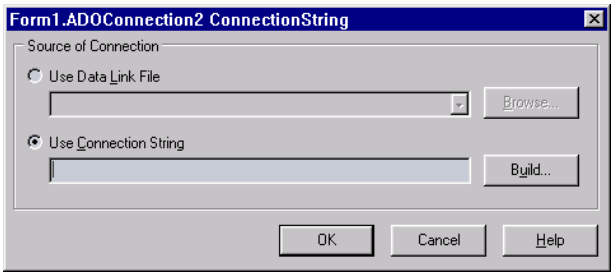
ADOQuery - компонент ADOQuery позволяет направить запрос серверу базы данных. Обычно он используется, если данные, которые надо получить из базы данных, распределены по нескольким таблицам [если данные находятся в одной таблице, то рекомендуется использовать компонент ADOTable]:

Все компоненты должны связываться с базой данных. Делается это двумя способами либо через компонент **TADOConnection** либо прямым указанием базы данных в остальных компонентах. К **TADOConnection** остальные компоненты привязываются с помощью свойства **Connection**, к базе данных напрямую через свойство **ConnectionString**.

База данных может быть указана двумя способами через файл линка к данным (файл в формате Microsoft Data Link, расширение UDL), либо прямым заданием параметров соединения.

Значение свойства всех **ConnectionString** этих компонент могут быть введены напрямую в текстовой форме, но куда проще вызвать редактор свойства нажав на кнопку «…» в конце поля ввода.

Окно этого свойства выглядит так:



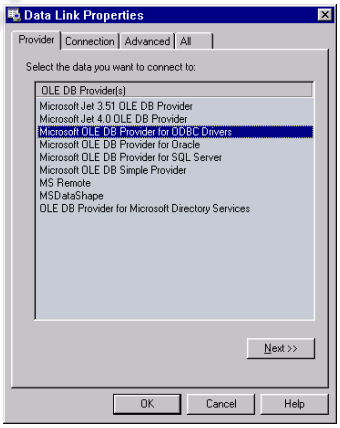
При выборе «Use data link file» и нажатии на кнопку «Browse…» появляется стандартный диалог выбора файла. Этот файл можно создать в любом окне explorer-а (в этом окне открытия файла, в самом explorer, на desktop и т.д.) вызвав контекстное меню и выбрав пункт «New/Microsoft Data Link». Потом вызовите локальное меню для созданного файла и выберите в нем пункт «Open». После этого появится property sheet описанный чуть ниже. Эти же вкладки содержит и property sheet, вызываемый через пункт «Property» локального меню UDL файла, но в нем еще есть вкладки относящиеся к самому файлу.

Использование файлов Microsoft Data Link упрощает поддержку приложений, так как возможно использовать средства Windows для настройки приложения.

При выборе в редакторе свойства «Use connection string» и нажатии на кнопку «Build…» появляется такой же property sheet, как и при выборе «Open» для Microsoft Data Link файла.

В этом окне выбирается тип базы данных, местоположение базы и параметры соединения.

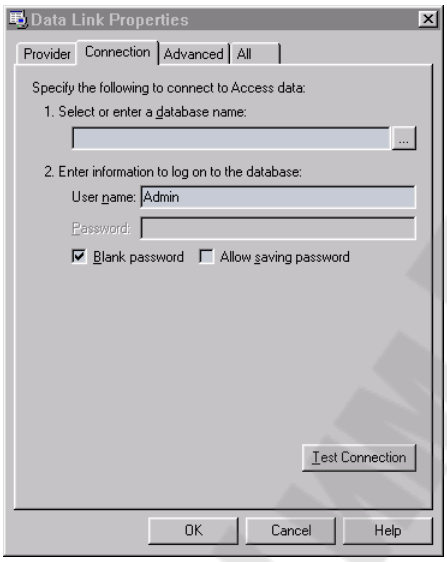
На первой странице выбирается тип базы данных или Provider, в терминах ADO.



Базы MS Access доступны как через «Microsoft Jet OLE DB Provider», так и через «Microsoft OLE DB Provider for ODBC».

Следующая страница зависит от выбранного типа базы, однако для всех типов есть кнопка «Test connection» позволяющая проверить правильность и полноту параметров.

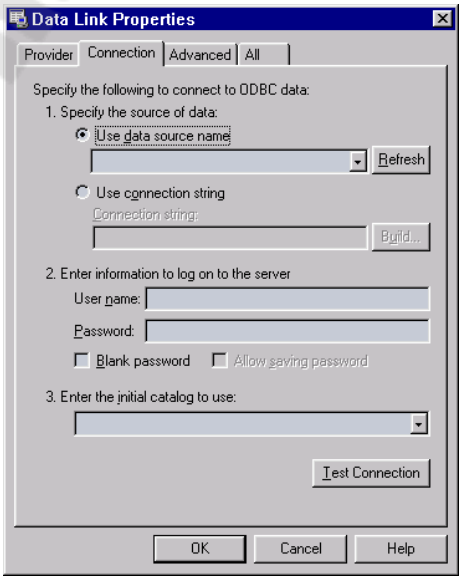
Для «Microsoft Jet OLE DB Provider» она выглядит так:



Checkbox «Blank password» подавляет диалог ввода идентификатора и пароля пользователя при установлении соединения, если поле пароля пустое.

Checkbox «Allow saving password» сохраняет пароль в строке параметров соединения. Если он не отмечен, то введенный пароль будет использоваться только при выполнении тестового соединения.

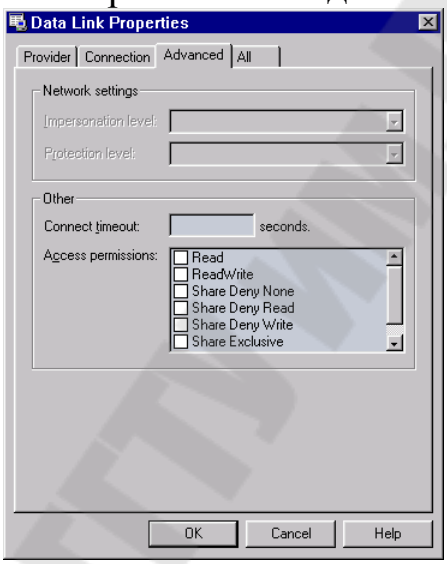
Для «Microsoft OLE DB Provider for ODBC» эта страница выглядит так:



Радиокнопка «Use data source name» позволяет ввести предустановленный алиас ODBC, а через «Use connection string» вводится как алиасы так и тип ODBC драйвера и параметры соединения.

Параметры идентификации пользователя аналогичны выше описанным.

На странице «Advanced» расположены дополнительные параметры, с помощью которых устанавливается уровень доступа к файлу базы данных, таймаут сетевого соединения (то есть время через которое связь будет считаться потерянной, если сервер не отвечает) и уровень глубины проверки секретности соединения.



На странице «All» можно отредактировать все параметры с предыдущих страниц и параметры зависящие от провайдера, но не вошедшие на страницу «Connection». Редактирование осуществляется в виде параметр – значение, причем в текстовой форме, никаких диалогов нет. Помощи то же нет, эти параметры описаны только в документации на провайдер. Ее можно найти в MSDN Data Access Services/Microsoft Data Access Components (MDAC) SDK/Microsoft ActiveX Data Objects (ADO)/Microsoft ADO Programmer's Reference/Using Providers with ADO.

В компоненте **TADOConnection** есть свойства **Provider**, **DefaultDatabase** и **Mode** которые являются альтернативным методом задания частей строки параметров соединения – провайдера, базы данных (например, пути до базы MS Access) и режима совместного использования файлов базы данных. Эти значение этих свойств автоматически включаются в строку соединения, если были заданы до

активизации компонента и автоматически выставляются после соединения.

Задание

Для заданной предметной области разработать приложение, работающее с БД с помощью технологии ADO. Базу данных разработать в MS Access. Взаимодействие с БД обеспечить с помощью компонент TADOConnection, TADOQuery, TADOTable.

Контрольные вопросы

1. Особенности технологии ADO.
2. Свойства компоненты TADOConnection.
3. Свойства компоненты TADOQuery.
4. Свойства компоненты TADOTable.

Лабораторная работа № 8

Тема: Формирование отчетов

Цель работы: ознакомиться со способами создания визуальных приложений, способных формировать предварительный просмотр и печать отчета данных из БД.

Теоретические сведения

Отчеты предназначены для быстрого получения визуальной информации из данных.

Для решения данной проблемы, на наиболее информационный манер, традиционные генераторы отчетов предлагают, секционное, в стиле таблиц представление данных.

На сегодняшний день имеются более сложные требования к отчетам, которые не так легко решить стандартными секционно-табличными средствами.

Визуальный Rave дизайнер предлагает многие уникальные свойства, которые помогут сделать процесс получения отчетов проще, быстрее и более понятным.

Rave является интуитивной, визуальной средой разработки, которая может легко управлять широким спектром отчетов, много больше, чем чистый, рассчитанный на секции дизайнер.

Rave также включает зеркальное отражение и другие технологии, для повторного использования содержимого ваших отчетов для быстрого внесения изменений и боле простого обслуживания.

В действительности, Rave был специально разработан для получения гибкости и функциональности, в простом для освоения формате.

В инсталляцию Rave включен проект, названный "RaveDemo", который содержит примеры различных отчетов.

Имеются два типа объектов в Rave, компоненты вывода (Output Components) и классы отчета (Report Classes). Компоненты вывода отвечают за вывод отчета на различные устройства вывода, а классы отчета, которые не являются компонентными классами, отвечают за все остальные задачи.

Компонент TRvProject является ключом для доступа к визуальным отчетам, создаваемым с помощью Rave. Обычно у вас только один компонент TRvProject на все приложения, но при нужде Вы можете иметь их столько, сколько нужно. Свойство ProjectFile определяет файл проекта вашего приложения, в котором хранятся все определения отчета. Данный файл имеет расширение .RAV и даже если это единственный файл он может хранить столько определений отчета сколько необходимо.

Когда вызывается метод Open объекта TRaveReport, то данный файл загружается в память для подготовки к печати или для изменений пользовательским дизайнером. Вы должны обязательно вызвать метод Close, как только вам не нужен файл проекта или при закрытии вашего приложения. Любые изменения в проекте отчета Вы можете сохранить, вызвав метод Save. TRvProject также имеет несколько свойства и методов, такие как SelectReport, GetReportList, ReportDescToMemo, ReportDesc, ReportName и ReportFullName, что делает эффективным и простым создание интерфейса для ваших пользователей.

Задание

На основании проекта лабораторной работы № 7 разработать приложение, обеспечивающее формирование предварительного просмотра и печати отчета данных из БД.

Пояснения к выполнению работы

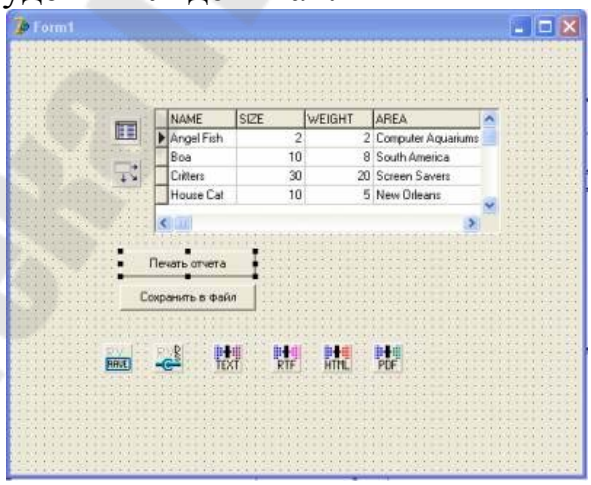
Созданные отчеты Rave можно сохранять в удобном формате. Это позволяет экспортировать и открывать их из других программ (Adobe Reader, Internet Explorer, MS Word...) и в случае форматов .rtf и .txt редактировать.

Для начала создадим новый проект. Скинем на форму Ttable, TDataSource, TDbGrid. Свяжем их с таблицей Animals.dbf Создаваемый отчет будет содержать информацию из таблицы Table1.

Для работы с Rave необходимо перенести на форму следующие компоненты:

для вывода и печати отчета TRvProject; TRvDataSetConnection; для печати отчета в другие форматы: TRvNDRWriter; TRvRenderText; TRvRenderRTF; TRvRenderHTML; TRvRenderPDF.

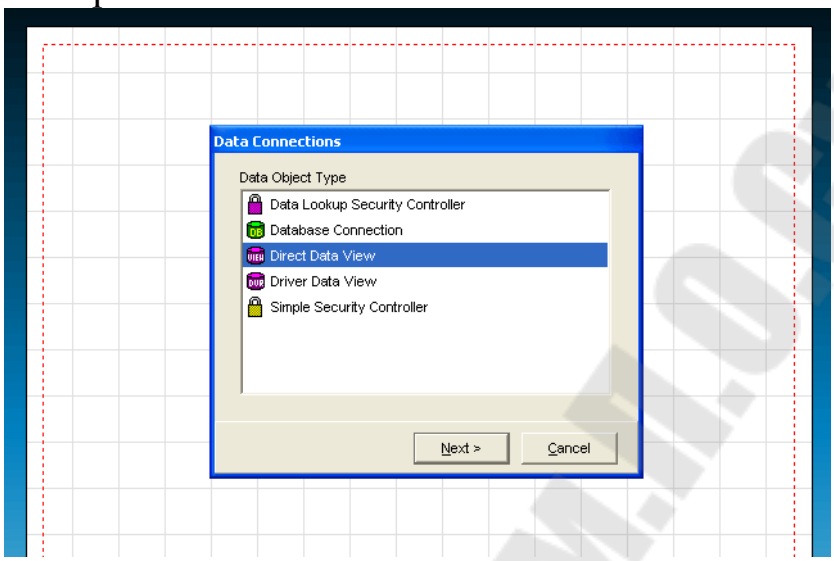
Печать отчета и сохранение в отдельный файл будем осуществлять при нажатии на соответствующую кнопку. В результате полученная форма будет выглядеть так:



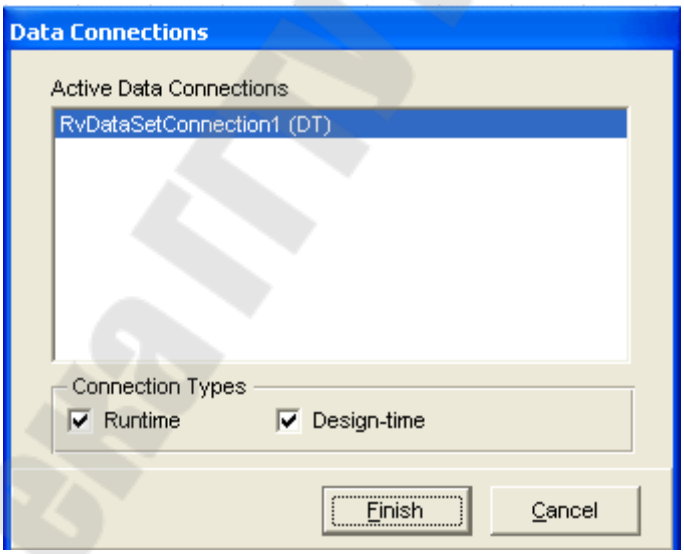
Теперь присвоим RvDataSetConnection1 свойство Dataset значение table1. Таким образом в Rave будут передаваться данные из Table1.

Переходим в Rave Designer (Tools Rave Designer).

Автоматически будет создан новый проект. Свяжем проект с Table1 через RvDatasetConnection1. Для этого нажмите кнопку New Data Object и выберите Direct Data View



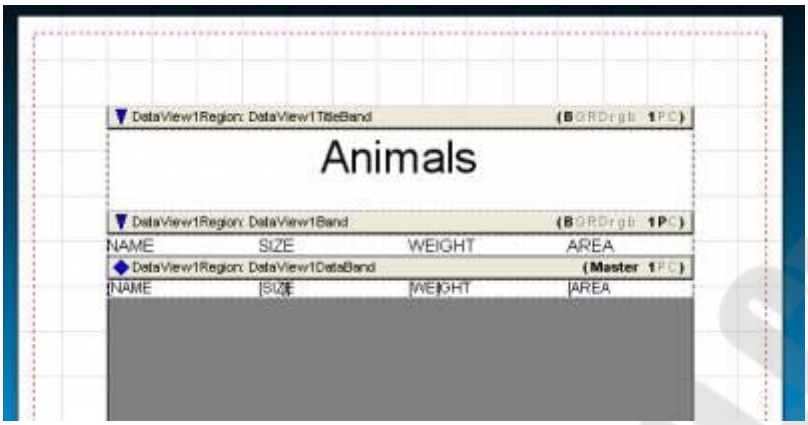
Нажимаем Next и в появившемся окне должны быть отражены все активные Rave соединения. В нашем случае это будет выглядеть так



После нажатия Finish справа в Rave Designer в Data View Dictionary появится Dataview1, раскрыв который можно увидеть поля Table1.

Теперь создадим быстро простой отчет. Для этого воспользуемся Tools\Report Wizards\Simple Table.

Выберим Dataview1, выделим все поля и завершим создание. После небольшой корректировки полученный лист отчета будет выглядеть следующим образом



В верхнем поле (TitleBand) заголовок страницы, ниже заголовки колонок и в последней помещаются DataText Component, которые будут принимать значения соответвующего поля (колонки).

Загляните в Report Library (справа) и посмортите, что бы имя текущего листа отчета было Report1. Сохраним полученный проект в папку с проектом Delphi. Получится файл Project1.rav.

Теперь вернемся к нашему проекту в Delphi и напишем обработчик события для кнопки печать отчета. Соединим RvProject1 с полученным файлом Project1.rav. Можно сделать это программно. И далее запустим наш отчет.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
RvProject1.SetProjectFile(ExtractFilePath(Application.ExeName)+'Project1.rav');
RvProject1.ExecuteReport('Report1');
RvProject1.Close;
end;
```

Теперь в результате нажатия на кнопу мы увидим диалоговое окно, в котором можно выбрать печатать отчет на принтере, посмотреть перед печатью или сохранить в файл.

Уже сейчас можно выбрать и сохранить в файл в удобном для нас формате.

Далее рассмотрим как сделать это напрямую без диалогового окна через программный код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NdrStream: TMemoryStream;
  OutStream: TMemoryStream;
  name:string;
begin
  NdrStream := TMemoryStream.Create;
  OutStream := TMemoryStream.Create;
```

```
  RvProject1.Close;
  RvProject1.SetProjectFile(ExtractFilePath(Application.ExeName)+'Project1.rav');
  RvProject1.SelectReport('Report1',true);
  try
    RvNDRWriter1.StreamMode := smUser;
    RvNDRWriter1.Stream := NdrStream;
    RvProject1.Engine := RvNDRWriter1;
    RvProject1.Execute;
    name:='Report1';
    RvRenderText1.PrintRender(NdrStream,name+'.txt');
    RvRenderRTF1.PrintRender(NdrStream,name+'.rtf');
    RvRenderHTML1.PrintRender(NdrStream,name+'.html');
    RvRenderPDF1.PrintRender(NdrStream,name+'.pdf');
  finally
    FreeAndNil(NdrStream);
    FreeAndNil(OutStream);
  end;
  RvProject1.Close;
end;
```

В результате нажатия на кнопку будут созданы 4 файла в папке с программой: Report1.txt, Report1.rtf, Report1.html, Report1.pdf.

Контрольные вопросы

1. Принципы создания отчётов. Компоненты для построения отчётов.
2. Создание отчёта. Мастер создания отчётов.
3. Изменение отчёта.
4. Группирующий отчёт.

Лабораторная работа № 9

Тема: Использование Com технологий в Delphi

Цель работы: получить навыки создания контроллеров автоматизации приложений MS Office (MS Word, MS Excel) для документирования информации, содержащейся в базах данных.

Теоретические сведения

Работа с MS Office

Так как многие разрабатываемые приложения формируют множество выходных отчетных форм для печати, то возможность формирования ими документов непосредственно в формате Word или Excel выгодно отличало бы их от своих аналогов.

Чтобы работать с MS Office из Delphi существует два основных пути.

- Первый способ (для таблицы Excel) –подключение через ADO, DAO(Data Active Objects) или ODBC.
- Второй способ – это создание его как COM объект.

Через ODBC/ADO

Это не самый быстрый и далеко не первый по возможностям метод (DAO работает на порядок быстрее и предоставляет куда больше возможностей).

1. заходим в Панель управления Windows, идем в свойства ODBC, делаем DSN, используя Excel драйвер, не забываем указать в свойствах на файл Excel. Закрываем ODBC.
2. Ставим на форму ADOConnection. В ConnectionString - строим строку подключения - надо выбрать только ODBC провайдер и на следующей вкладке указать сделанный DSN. Ее можно вообще упростить до вида: "DSN=MyDsn". Теперь вам доступны листы файла как таблицы, а весь файл как база данных.
3. Подключаем ADOQuery к ADOConnection. Создаем таблицу, т.е. новый лист:

   ```
   Create Table MyTable1 (
   Field1 varchar(20),
   Field2 varchar(10) )
   ```

4. Снова переходим в дизайн - ставим на форму ADOTable, указываем как Connection наш компонент с ADOConnection, теперь если кликнуть на свойстве TableName - вы сможете увидеть в списке сделанную нами таблицу "MyTable1".

5. Соедините таблицу с DBGrid. Работа с таблицей в Excel мало отличается от работы с другими базами данных.

Com соединения

Текстовый редактор Word представляет собой COM-сервер и может получать и обрабатывать запросы от внешних программ. Все это позволяет организовать процесс управления и создания документа из внешней программы. Используя этот механизм, можно создать документ программно так же, как это делается вручную (посредством меню, кнопок и клавиатуры), но гораздо быстрей и эффектней.

Word предоставляет три объекта, через которые можно получить доступ к внутренним объектам Word-а и документов.

Word.Application,
Word.Document,
Word.Basic.

Ко всем остальным объектам (текст, таблицы, кнопки, меню и др.) доступ возможен только через них.

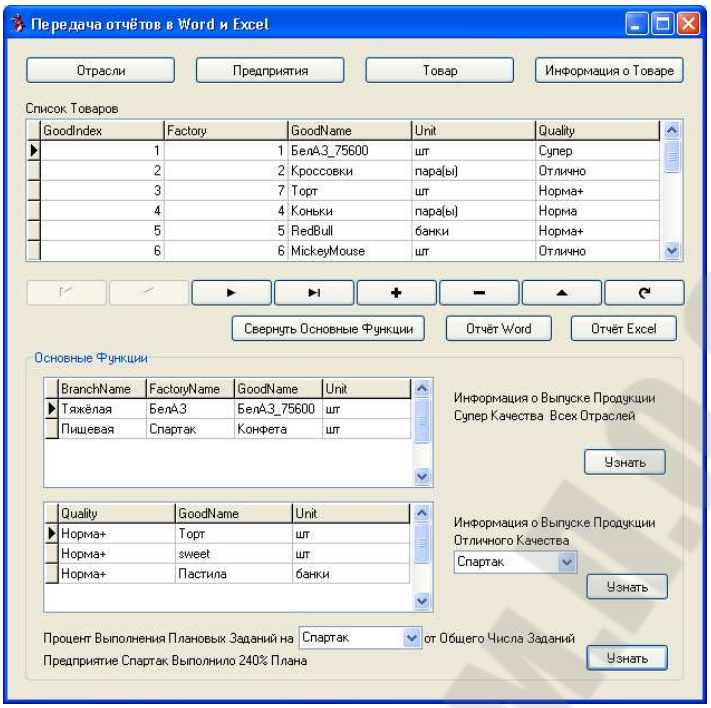
Задание

в соответствии с заданием на лабораторную работу № 8 произвести выборку данных из разработанной базы данных, полученные данные передать в MS Word и MS Excel для формирования отчетов и построения диаграммы.

Пояснения к выполнению работы

Имеется разработанное приложение для работы с базой данных конкретной предметной области. Доступ к базе данных осуществляется через Microsoft ActiveX Data Objects (ADO). Необходимо дополнить приложение возможностью передачи информации из базы данных в MS Word и MS Excel для формирования отчетов и построения диагрраммы.

1. Добавляем в макет приложения две кнопки.



2. В главном модуле приложения подключаем модули ComObj и ActiveX

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComObj, ActiveX;
```

3. Для обработчика события нажатия кнопки «Отчёт Word» записываем следующий код

```
procedure TForm1.Button10Click(Sender: TObject);
var
  i: integer;
  App:Variant;
  table,M,Sel: Variant;
begin

  //Создание контроллера
    App:=CreateOleObject('Word.Application');
  App.Visible:=true;
  App.Documents.Add;

  // Формирование заголовка
   App.ActiveDocument.Paragraphs.Item(1).Range.Select;
   Sel := App.Selection;
   Sel.Font.Bold := True;
   Sel.TypeText('Отчет о выпуске продукции на предприятии "' + ComboBox1.Text + '"');
   Sel.TypeParagraph;
   Sel.Font.Bold := False;
   Sel.TypeParagraph;
```

```
 //Создание таблицы
 M:=App.ActiveDocument.Range(App.ActiveDocument.Range.End-1, App.ActiveDocument.Range.End-1);
 App.ActiveDocument.Tables.Add(M, ADOQuery1.RecordCount+1, 3);
 App.ActiveDocument.Range.InsertAfter(' ');
 ADOQuery1.First();

 //Создание заголовка таблицы
  table:= App.ActiveDocument.Tables.Item(1).Cell(1,1).Range;
  table.InsertAfter('Качество');
  table:= App.ActiveDocument.Tables.Item(1).Cell(1,2).Range;
  table.InsertAfter('Название Товара');
  table:= App.ActiveDocument.Tables.Item(1).Cell(1,3).Range;
  table.InsertAfter('Единица Измерения');

 //Перенос данных из Delphi в Word
 i:=2;
while not ADOQuery1.Eof  do
 begin
  table:=  App.ActiveDocument.Tables.Item(1).Cell(i,1).Range;
  table.InsertAfter(ADOQuery1.FieldByName('Quality').AsString);
  table:=  App.ActiveDocument.Tables.Item(1).Cell(i,2).Range;
  table.InsertAfter(ADOQuery1.FieldByName('GoodName').AsString);
  table:=  App.ActiveDocument.Tables.Item(1).Cell(i,3).Range;
  table.InsertAfter(ADOQuery1.FieldByName('Unit').AsString);
   i:=i+1;
 ADOQuery1.Next();
end;

 //Форматирование ячеек по содержимому
 App.ActiveDocument.Tables.Item(1).Columns.AutoFit;

 // Выравнивание текста по центру
 App.ActiveDocument.Range.ParagraphFormat.Alignment:=1;

 // Включение Отображения Таблицы
 App.Visible:=true;

 App.ActiveDocument.Range.InsertAfter(#10 + #13 + #10 + #13 + 'Пример формирования отчета');

 //Сохранение документа и закрытие сервера Word
 App.ActiveDocument.SaveAs('C:\Отчет_Word.doc');
 App.Application.Documents.Close;
 App.Quit;
 end;
```

4. Для обработчика события нажатия кнопки «Отчёт Word» записываем следующий код

```
procedure TForm1.Button11Click(Sender: TObject);

const
  xl3DColumn = -4100;
  xl3DBar = -4099;

var
    Ch, Rng, Sel     : Variant;
    i                : Integer;
begin
  // Создать один экземпляр сервера
  App := CreateOleObject('Excel.Application');
  // показать окно приложения на экране
  App.Visible := True;
  App.Workbooks.Add();
  I:=App.Workbooks.Count;
  App.Workbooks[I].Activate;
  //Обращение к страницам
  App.ActiveWorkbook.WorkSheets[1].Name := 'Отчет Excel';
  // Формирование Заголовка таблицы
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[2,2].Value := 'Качество';
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[2,3].Value := 'Название Товара';
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[2,4].Value := 'Единица Измерения';
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[2,5].Value := 'Количество';

// Перенос данных из Delphi в Excel
   Randomize;
   ADOQuery1.First();
   i:=2;
 while not ADOQuery1.Eof  do
  begin
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[i+1,2].Value := ADOQuery1.FieldByName('Quality').AsString;
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[i+1,3].Value := ADOQuery1.FieldByName('GoodName').AsString;
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[i+1,4].Value := ADOQuery1.FieldByName('Unit').AsString;
  App.ActiveWorkbook.WorkSheets['Отчет Excel'].Cells[i+1,5].Value := Random(100);
    i:=i+1;
  ADOQuery1.Next();
end;
  //Форматируем ячейки
   App.ActiveWorkbook.WorkSheets[1].Range['A2:E5'].Font.Size := 12;
   App.ActiveWorkbook.WorkSheets[1].Range['A2:E2'].Font.Bold := True;
   App.ActiveWorkbook.WorkSheets[1].Range['A2:E5'].Select;
   Sel := App.Selection;
   Sel.ColumnWidth:=26;

  //Строим график
   Ch := App.ActiveWorkbook.WorkSheets[1].ChartObjects.Add(50,100,450,230);
   Ch.Chart.ChartWizard(App.ActiveWorkbook.WorkSheets[1].Range['E2:E5'],xl3dColumn);
   Ch.Chart.HasTitle               := 1;
   Ch.Chart.HasLegend              := False;
   Ch.Chart.ChartTitle.Text        := 'Пример диаграммы Excel';
   Ch.Chart.Axes(1).HasTitle       := True;
   Ch.Chart.Axes(1).AxisTitle.Text := 'Товар';
   Ch.Chart.Axes(2).HasTitle       := True;
   Ch.Chart.Axes(2).AxisTitle.Text := 'Количество';

   //Сохраняем файл
   App.DisplayAlerts:=False;
   App.ActiveWorkBook.SaveAs('C:\Отчет_Excel.xls');
   //Закрываем рабочую книгу
   App.ActiveWorkBook.close;
   //Закрываем Excel
   App.Quit;
   App:=Unassigned;

end;
```

5. Запускаем приложение для проверки работоспособности.

Контрольные вопросы

1. В каких модулях находятся функции Delphi для работы с Com серверами.
2. Как создать экземпляр сервера автоматизации.
3. Как закрыть сервер автоматизации.
4. Как создать документ в MS Word.
5. Как создать рабочую книгу в MS Excel.
6. Как создать новый абзац в MS Word.
7. Как вставить текст в MS Word.
8. Как отформатировать текст в MS Word.
9. Как вставить таблицу в документ MS Word.
10. Как сохранить документ MS Word.
11. Как отформатировать содержимое таблицы.
12. Как передать данные в рабочую книгу в MS Excel
13. Как отформатировать таблицу в MS Excel.
14. Как вставить диаграмму в MS Excel.
15. Как отформатировать диаграмму в MS Excel.
16. Как сохранить рабочую книгу MS Excel.

Литература

1. Гамильтон Б. ADO.NET Сборник рецептов. Для профессионалов. – СПб.: Питер, 2005. – 576 с.

2. Фаронов В.В. Программирование баз данных в Delphi 7. Учебный курс. – СПб.: Питер, 2006. – 459 с.

3. Архангельский А.Я. Приемы программирования в Delphi на основе VCL. – М.: Бином-Пресс, 2006. – 944 с.

4. Архангельский А.Я. Справочное пособие: язык Delphi, классы, функции Win32 и .NET. – М.: Бином-Пресс, 2006. – 1152 с.

5. Шпак Ю.А. Delphi 7 на примерах. – К.: Юниор, 2003. – 384 с.

6. Дарахвелидзе П.Г., Марков Е.П. Программирование в Delphi 7. – СПб.: БХВ-Петербург, 2003. – 784 с.

7. Бакнелл Дж.М. Фундаментальные алгоритмы и структуры данных в Delphi / Пер. с англ. – СПб.: ДиаСофтЮп, 2003. – 560 с.

Содержание

Рябченко Александр Иванович
Самовендюк Николай Владимирович
Емельянченко Наталья Сергеевна

СРЕДСТВА ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ

**Лабораторный практикум
по одноименной дисциплине для слушателей
специальности 1-40 01 73 «Программное обеспечение
информационных систем»
заочной формы обучения**