Министерство образования и науки Челябинской области

**государственное бюджетное образовательное учреждение**

**среднего профессионального образования**

**(среднее специальное учебное заведение)**

**«Южно-Уральский многопрофильный колледж»**

**Методические указания**

**по выполнению практических работ**

***МДК 03.01 «Технология разработки программного обеспечения»***

***ПМ.03 «Участие в интеграции программных модулей»***

## г. Челябинск, 2015

**ОДОБРЕНА УТВЕРЖДАЮ**

Цикловой методической комиссией Заместитель директора

ЕН дисциплин, ОПД и ПМ \_\_\_\_\_\_\_\_\_И.Н.Тихонова

 Специальности «\_\_\_»\_\_\_\_\_\_\_\_\_2015 г.

«Программирование в

компьютерных системах»

Председатель ЦК

­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_Сухорослова Л.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2015 г.

Методические указания для организации практических работ и подготовки курсового проекта для МДК.03.01 «Технология разработки программного обеспечения» разработана на основе Федерального государственного образовательного стандарта (ФГОС) по специальности среднего профессионального образования **09.02.03 Программирование в компьютерных системах**, утвержденного приказом Министерства образования и науки РФ 21.08.2014 № 33733.

Организация-разработчик:

государственное бюджетное образовательное учреждение среднего профессионального образования (среднее специальное учебное заведение) «Южно-Уральский многопрофильный колледж»

Согласовано:

\_\_\_\_\_\_\_\_\_\_\_Старший методист, Кулиненко И.М.

Разработчик:

**Сухорослова Людмила Валентиновна,** преподаватель ЮУМК

Рецензент:

**Сальникова Надежда Павловна**, начальник отдела обработки данных СИТ АО ЧМК.

**Тема 1: «Компоненты доступа и представления данных»**

Для представления данных в приложении VCL предусмотрено 2 группы компонентов - Data Access и Data Controls.

Для доступа к данным, представленным при помощи различных компонент - будь то BDE-ориентированные источники (например, Table), или ADO, IB Express, или dbExpress, используется один и тот же набор компонентов, расположенных на закладке Data Access:

* DataSource - источник данных;
* ClientDataSet - клиентский набор данных;
* DataSetProvider - провайдер набора данных;
* XMLTransform - преобразователь данных, представленных в виде XML в обычный пакет данных и обратно;
* XMLTransformProvider - провайдер данных для XML-документов, осуществляющий так же их обновление;
* XMLTransformClient - адаптер между XML-документом и провайдером.

Из этого списка нам интересно только первые 3 компонента, а именно DataSource, ClientDataSet и DataSetProvider. Используя набор из этих компонент, можно обеспечить доступ к данным. Причем в случае, когда речь идет о BDE и таблицах Paradox, как правило, достаточно использовать лишь один из перечисленных компонентов - DataSource. Этот компонент имеет всего 4 собственных свойства - AutoEdit, DataSet, Enabled и State. Свойство Enabled похоже на свойство Active таблицы или Connected у базы данных, т.е. делает активным или неактивным соединение. А свойство AutoEdit, будучи включенным, обеспечивает возможность правки записей без написания какого-либо дополнительного кода. Свойство State информирует о том, в каком состоянии в текущий момент находится источник данных. Ну а самое важное свойство этого компонента - это, конечно же, DataSet, которое и определяет источник данных - таблицу, запрос и т.д.

Оставшиеся ClientDataSet и DataSetProvider могут понадобиться в том случае, если требуется обеспечить кэширование записей, например, для того, чтобы представить в виде таблицы источник данных типа dbExpress.

Рассмотрим пример, когда нам требуется обеспечить доступ к данным простой таблицы Paradox. Для этого нам на форме приложения понадобятся следующие компоненты: Database и Table из BDE, а так же DataSource из Data Access.

ПРИМЕЧАНИЕ
Хотя для таблицы (компонента Table) можно указать один из определенных в BDE псевдонимов без помощи компонента Database, по сложившейся традиции, а так же в целях удобства управления приложением, все-таки предпочтительнее использовать связку из Database и Table.

Теперь для свойства AliasName компонента Database выберем название имеющегося у нас псевдонима БД (DATA1), а в качестве значения свойства DatabaseName так же напишем DATA1. Таким образом, компонент Database будет видеть "настоящий" псевдоним DATA1, а все остальные компоненты приложения - псевдоним DATA1, определенный посредством компонента Database.

Если такое положение вещей вас смущает, то в качестве значения DatabaseName можно указать любое произвольное значение, например, MyDatabase - в таком случае у других компонентов, имеющих свойство DatabaseName среди возможных значений этого свойства, будет значиться и MyDatabase.

Тем не менее, мы остановимся на начальном варианте, и перейдем к компоненту Table, для которого нам так же придется установить значения для 2 свойств - DatabaseName и TableName. Для первого укажем DATA1, для второго - customer. Чтобы убедиться, что все сделано правильно, попробуем активировать связь с БД, для чего установим в истину свойство Active. Если все было сделано верно (включая создание псевдонима DATA1 и таблицы customer, о чем речь шла в предыдущей главе), то не только свойство Active таблицы изменится на истину, но и свойство Connected компонента Database так же изменится на истину.

Последнее, что осталось сделать - это поместить на форму компонент DataSource и установить значение его свойства DataSet в Table1. Таким образом, мы получим действующую связку из 3 компонентов, обеспечивающих все этапы взаимодействия с БД - от организации локального псевдонима и управления им (Database), до выбора конкретной таблицы с возможностью управления ее параметрами (Table) и предоставления ее данных любым другим компонентам (DataSource).

**Тема 2: Таблица DB Grid**

Теперь настало время рассмотреть собственно компоненты, которым могут понадобиться данные для представления. Все они расположены на закладке Data Controls. Прежде всего, это, конечно, специальная таблица для баз данных - DBGrid. Этот компонент является дальнейшим развитием обычной таблицы (StringGrid), но предназначен исключительно для отображения и редактирования связанной с БД информации. Соответственно, у DBGrid нет таких свойств, как Cells, Cols и Rows, поскольку все, что выводит этот компонент - есть прямое отражение текущего содержимого связанной с ним таблицы БД.

В то же время, у компонента DBGrid предусмотрен целый ряд специальных свойств, предназначенных для взаимодействия с БД. Прежде всего, это свойство DataSource, в котором указывают имя компонента-источника данных. Так, если на форму, где уже имеются настроенные соответствующим образом невизуальные компоненты Database, Table и DataSource поместить таблицу, в свойстве DataSource которой указать DataSource1, то мы сразу же увидим содержимое таблицы customer (рис. 19.1).



Рис. 1. Форма Delphi с таблицей БД

Следует сразу же отметить, что в качестве заголовков столбцов были использованы названия полей таблицы БД. Кроме того, можно увидеть, что столбец, содержащий числовые данные, имеет выравнивание по правому краю, а строковые - по левому. Таким образом, очевидно, что компонент DBGrid имеет более широкие возможности по оформлению таблиц, чем обычная таблица StringGrid. Возможно это благодаря другому свойству DBGrid - Columns, которое определяет оформление, количество и порядок следования столбцов с данными. Это свойство представляет собой коллекцию, состоящую из отдельных колонок таблицы. По умолчанию используется автоматический режим вывода, когда выводятся все поля данных с размерами, основанными на параметрах самих полей, заданных в выводимой таблице БД. Но поскольку во многих случаях выводить все поля не требуется, или же необходимо изменить параметры их вывода, как-то порядок следования, цвет, шрифт, или ширину поля, то все эти настройки доступны именно через свойство Columns. При этом каждый элемент этого свойства представляет собой объект типа TColumn, имеющий, в свою очередь, такие свойства, как заголовок (вместо стандартного названия поля в БД) выравнивание, цвет фона, шрифт, возможность правки и т.д. Доступ ко всем этим параметрам возможен через специальный редактор коллекций, который можно вызвать, дважды щелкнув по самой таблице в режиме разработки (рис. 19.2).



Рис. 2. Редактирование списка столбцов

На панели инструментов редактора столбцов имеется 4 кнопки - для добавления и удаления столбцов, а так же для автоматического заполнения и для сброса к начальным установкам. Выбирая поле из списка, и изменяя его свойства в инспекторе объекта, вы тем самым изменяете параметры отображения соответствующего столбца. А меняя в редакторе строки местам (путем перетаскивания мышкой), вы меняете порядок вывода полей в самой таблице.

Таким образом, мы рассмотрели 2 наиболее важных свойства компонента DBGrid. Что касается всех собственных свойств, имеющихся у таблицы для баз данных, то они перечислены в таблице 19.1.

|  |
| --- |
| Таблица 19.1. Собственные свойства компонента DBGrid |
| **Свойство** | **Тип** | **Описание** |
| Columns | TDBGridColumns | Задает параметры вывода столбцов с данными |
| DataSource | TDataSource | Определяет источник данных для отображения в таблице |
| FieldCount | Integer | Указывает на число столбцов с данными, выводимых в таблице |
| Fields | array of TField | Предоставляет доступ к информации ячейки, находящейся в указанном столбце |
| Options | TDBGridOption | Определяет различные параметры отображения и поведения таблицы |
| ReadOnly | Boolean | Определяет, будет ли у пользователя возможность править данные в таблице |
| SelectedField | TField | Предоставляет доступ к информации в выделенной ячейке |
| SelectedIndex | Integer | Определяет номер текущего столбца |
| TitleFont | TFont | Определяет шрифт, используемый для вывода заголовков столбцов таблицы |

Здесь следует отдельно выделить свойство Options, позволяющее настроить целый ряд различных параметров. Оно имеет следующие флаги:

* dgEditing - Делает возможной правку данных прямо в таблице. Этот флаг игнорируется, если включен флаг dgRowSelect;
* dgAlwaysShowEditor - Таблица будет постоянно находиться в режиме редактирования. В противном случае пользователь должен будет нажимать F2, Enter, или щелкать мышкой по полю, чтобы ввести новое значение;
* dgTitles - Делает видимыми заголовки столбцов;
* dgIndicator - Добавляет колонку, в которой будет отображаться индикатор выбранной записи;
* dgColumnResize - Делает возможным изменение размеров столбцов пользователем;
* dgColLines - Столбцы будут отделены разделительными линиями;
* dgRowLines - Записи будут отделены разделительными линиями;
* dgTabs - Делает возможной навигацию по ячейкам при помощи клавиш Tab и Shift+Tab;
* dgRowSelect - Записи будут выделяться целиком. При этом правка данных в таблице становится невозможной (т.е. флаги dgEditing и dgAlwaysShowEditor будут проигнорированы);
* dgAlwaysShowSelection - Выбранная ячейка будет выделена цветом даже если фокус вводе не находится на таблице;
* dgConfirmDelete - Будет выдаваться предупреждение, если пользователь захочет удалить запись в таблице (при помощи Ctrl+Delete);
* dgCancelOnExit - Предотвращает запись пустых записей;
* dgMultiSelect - Делает возможным выбирать несколько записей одновременно (с использованием клавиши Ctrl).

Например, если таблица должна будет использоваться лишь для навигации по БД и просмотра значений, то будет рациональным установить флаг dgRowSelect. С одной стороны, это автоматически отключит возможность правки и ввода данных непосредственно в самой таблице пользователем, а с другой - будет визуально выделять текущую запись, при этом достаточно наглядно показывая пользователю, что правка невозможна.

В типичном случае все производимые при разработке приложения настройки компонента DBGrid сводятся к тому, что, поместив его на форму, указывают связанный источник данных, после чего при помощи редактора определяют состав и вид столбцов данных. В случае при необходимости так же выставляют нужные значения для флагов в свойстве Options.

**Тема 3: Навигация по таблице данных**

Хотя в ряде случаев для обеспечения возможности навигации по таблице достаточно использовать лишь стандартные средства, которыми располагает компонент DBGrid, в ряде случаев бывает полезным предоставить пользователю более наглядный элемент управления для навигации по данным и для их правки. Более того, подобный компонент будет просто необходим, если для предоставления данных используется не таблица, а набор отдельных элементов, отображающих данные из одного конкретного поля.

Для этих целей предусмотрен специальный компонент - DBNavigator. С его помощью можно перемещаться по записям таблицы а так же выполнять операции типа вставки новой записи или подтверждения изменений. Внешне он представляет собой панель со следующими 10 кнопками:

* First - переход на первую запись в таблице;
* Prior - переход на предыдущую запись;
* Next - переход на следующую запись;
* Last - переход на последнюю запись;
* Insert - вставка новой записи перед текущей;
* Delete - удаление текущей записи с переходом на следующую;
* Edit - переводит источник данных в режим редактирования записи;
* Post - запись измененных данных из текущей записи в БД;
* Cancel - отмена изменений данных в текущей записи;
* Refresh - обновление данных в буфере источника.

Часть этих кнопок можно отключить, воспользовавшись свойством VisibleButtons. Еще одно свойство, влияющее на внешний вид компонента DBNavigator - это Flat. Установив его в истину, можно придать панели "плоский" вид. А при помощи свойства Hints можно задать пояснительный текст всплывающей подсказки для каждой кнопки.

Еще одно свойство - ConfirmDelete определяет поведение этого компонента: если для него установлено значение истины, то при попытке удаления записи (т.е. при нажатии на кнопку Delete) будет выдаваться соответствующее предупреждение.

Наконец, свойство DataSource, как и у всех других компонент представления данных БД, указывает на источник данных, связанных с данным компонентом.

Для примера добавим компонент навигации на форму с таблицей, подобной той, что изображена на рис. 19.1. Достаточно установить свойство DataSource компонента DBNavigator в то же значение, что и у одноименного свойства компонента DBGrid, чтобы получить связанно работающие компоненты. Например, при редактировании записей в таблице, состояние кнопок в навигационной панели будет изменяться в соответствии с возможными действиями (рис. 19.3).



Рис. .3. Таблица DBGrid и панель навигации

Пример можно найти в каталоге Demo\Part4\SimpleTable.

**Тема 4: Представление отдельных полей данных**

Как мы уже знаем, для отображения таблиц БД в целом, используется табличный же компонент - DBGrid. В том же случае, когда надо отобразить содержимое лишь отдельных полей данных, используют соответствующие компоненты - DBEdit, DBImage, DBCheckBox и т.д., в зависимости от типа данных, которые требуется отобразить в том или ином случае.

Все эти компоненты являются специализированными вариантами обычных компонент, не связанных с БД:

* DBText - аналог текстовой подписи Label;
* DBEdit - аналог однострочного редактора Edit;
* DBMemo - аналог многострочного редактора (блокнота) Memo;
* DBImage - аналог компонента для вывода изображений Image;
* DBListBox - аналог списка ListBox;
* DBComboBox - аналог раскрывающегося списка ComboBox;
* DBCheckBox - аналог переключателя CheckBox;
* DBRadioGroup - аналог группы исключающих переключателей RadioGroup;
* DBRichEdit - аналог редактора форматированного текста RichEdit.

Основным отличием ориентированных на применение совместно с базами данных компонент является наличие у них свойства DataSource, при помощи которого они связываются с источником данных. Еще одно свойство - DataField, как раз и указывает на то поле, которое должно отображаться в данном компоненте. В остальном по своим функциональным способностям они повторяют свои не ориентированные на БД аналоги.

Что касается вариантов использования, то, например, мы можем заменить таблицу в предыдущем примере на пару компонент - DBText для отображения номера и DBEdit для вывода и редактирования названия. Соответственно, мы можем заменить табличное представление данных представлением типа "форма".

Для этого, удалив таблицу и разместив на форме компоненты DBText и DBEdit, установим для них обоих свойство DataSource в значение DataSource1, после чего для свойства DataField у метки выберем значение CUST\_ID, а для этого же свойства у редактора - CUST\_NAME. Не помешает также разместить на форме еще и 2 обычные метки (Label), при помощи которых можно вывести текст, поясняющий, что за информация выводится в том или ином поле (рис. 19.4).



Рис. 4. Использование отличного от таблицы формата отображения данных

ПРИМЕЧАНИЕ
Здесь следует отметить тот факт, что данные могут автоматически преобразовываться в тот формат, который доступен компоненту. Например, целочисленное значение номера клиента было автоматически преобразовано в строку для вывода в DBText. Вместе с тем, возможности преобразований небезграничны, поэтому следует внимательно подбирать подходящие компоненты для отображения информации в каждом конкретном случае.

### Тема 5: Компонент DBCtrlGrid

Особняком от других компонентов для отображения информации баз данных стоит компонент DBCtrlGrid, не имеющий прямых аналогов среди "обычных" компонентов VCL. Он состоит из набора однотипных панелей и позволяет отображать данные в произвольной форме. При этом каждая панель является платформой, на которой размещены простые БД-компоненты - такие, как DBText, DBEdit, DBCheckBox и т.п. У всех этих компонентов будет общий источник данных, который задается централизованно в свойствах самого DBCtrlGrid. Соответственно, останется лишь выбрать поля, которые помещенные на DBCtrlGrid компоненты будут отображать.

Во время разработки определяют единственную панель, по образу которой будут созданы ее точные копии. Например, можно взять за основу форму вывода таблицы клиентов в виде DBText и DBEdit, поместив на нее DBCtrlGrid, на котором, в свою очередь, расположить метку с редактором. В результате после запуска приложения эти 2 компонента будут продублированы на всех видимых панелях сетки (рис. 19.5).



Рис. 5. Компонент DBCtrlGrid во время разработки (слева) и во время выполнения (справа)

Данные в таких панелях-ячейках могут располагаться как вертикально, так и горизонтально, либо вообще в качестве таблицы - в несколько рядов и колонок. Формат вывода панелей, как и их размеры, а так же ряд иных параметров этого компонента, определяется при помощи ряда свойств, перечисленных в таблице 19.2.

|  |
| --- |
| Таблица 2. Свойства компонента DBCtrlGrid |
| **Свойство** | **Тип** | **Описание** |
| AllowDelete | Boolean | Определяет, может ли пользователь удалить текущую запись, нажав Ctrl+Delete. |
| AllowInsert | Boolean | Определяет, может ли пользователь вставить новую запись, нажав клавишу Insert или добавить запись в конец, нажав Ctrl+Insert |
| ColCount | Integer | Определяет число столбцов с панелями |
| DataSource | TDataSource | Определяет источник данных для отображения |
| Orientation | TDBCtrlGridOrientation | Определяет порядок следования записей. Допустимые значения: goVertical (с вертикальной прокруткой) и goHorizontal (с горизонтальной прокруткой) |
| PanelBorder | TDBCtrlGridBorder | Определяет, должна ли быть рамка вокруг каждой панели. Допустимые значения: gbNone, gbRaised |
| PanelCount | Integer | Указывает на число видимых панелей |
| PanelHeight | Integer | Определяет высоту каждой панели в пикселях |
| PanelIndex | Integer | Определяет порядковый номе выбранной панели |
| PanelWidth | Integer | Определяет ширину каждой панели в пикселях |
| RowCount | Integer | Определяет число строк с панелями |
| SelectedColor | TColor | Определяет цвет фона активной панели |
| ShowFocus | Boolean | Определяет, должна ли отображаться дополнительная рамка вокруг панели при получении фокуса ввода |

Размеры панелей определяются при помощи свойств PanelHeight и PanelWidth, а размеры компонента в целом определяются так же количеством панелей по горизонтали и по вертикали, определяемые через свойства ColCount и RowRount.

ПРИМЕЧАНИЕ
Не рекомендуется помещать на панели ресурсоемкие компоненты вроде блокнота или изображения, особенно если панелей много. Так же следует учитывать, что все компоненты будут иметь один и тот же источник данных, определенный для самого DBCtrlGrid.

Пример с использованием данного компонента совместно с панелью навигации можно найти в каталоге Demo\Part4\CtrlGrid.

**Тема 6: Связывание данных в таблицах**

Важной особенностью работы с БД является связывание данных в таблицах. И хотя такое связывание осуществляется невизуальным компонентом Table, наглядно продемонстрировать связывание можно лишь с использованием компонентов, отображающих данные таблиц, т.е. с использованием, например, рассмотренного в этой главе компонента DBGrid.

Очевидно, что для связывания данных между таблицами нам нужно иметь более одной таблицы. Поэтому для начала откроем Database Desktop и создадим новую таблицу, предназначенную для связывания с уже имеющейся у нас таблицей клиентов.

Пусть это будет таблица счетов (Bill) для этих самых клиентов. Таким образом, в новой таблице нам понадобится, как минимум, 3 поля: номер счета (BILL\_ID), идентификатор клиента (BILL\_CUST) и сумма (BILL\_SUMM). При этом поле номера счета будет иметь автоинкрементный тип, и являться индексом, а поле с номером клиента - целочисленный тип и так же быть индексированным. Наконец, поле суммы будет денежного типа.

После создания структуры таблицы останется заполнить ее произвольными значениями - достаточно будет ввести по 1-2-3 записи для каждого из клиентов (рис. 19.6). Достаточно лишь помнить, что число, вводимое в поле идентификатора клиента должно соответствовать одному из значений в поле CUST\_ID таблицы клиентов, т.к. если за этим не следить, то получатся "потерянные" данные, не связанные с другими, что является нарушением целостности БД.



Рис. 6. Структура и содержимое таблицы счетов Bill

После подготовки исходных данных, перейдем к разработке приложения. Нам потребуется 1 компонент Database и по 2 компонента Table, DataSource и DBGrid. Для компонента Database установим те же значения, что и в предыдущем примере, т.е. AliasName и DatabaseName установим в DATA1, после чего свойство Connected можно установить в истину. Затем для первого "комплекта" из Table, DataSource и DBGrid так же устанавливаем значения, аналогичные предыдущему примеру, т.е. для Table1 установим свойство DatabaseName - в DATA1, а TableName - в customer.DB. Для DataSource1 установим свойство DataSet в Table1, и, наконец, для DBGrid1 установим свойство DataSource в DataSource1.

Затем подобную операцию следует провести для 2-го набора компонент, установив свойства подобным образом, за исключением того, что свойство TableName у компонента Table2 будет bill.DB, а во всех остальных случаях, разумеется, будут использованы имена этого набора (т.е. Table2, DataSource2, и т.д.). Если теперь для компонентов Table1 и Table2 установить значение свойства Active в истину, то в DBGrid1 и DBGrid2 мы увидим содержимое таблиц Customer и Bill.

Теперь займемся собственно связыванием данных. Для этого нам надо в подчиненной таблице, которую здесь представляет компонент Table2, указать на главную таблицу и сослаться на связанное поле, по которому и будет производиться отбор. Поскольку главная таблица предоставлена своим источником данных (DataSource1), то его и следует указать в свойстве MasterSource компонента Table2. Затем для свойства MasterFields установим значение CUST\_ID. Наконец, в свойстве INDEX\_NAME укажем вторичный индекс таблицы Bill, т.е. тот, который индексирует поле BILL\_CUST. В нашем случае он называется CUST\_IDX.

Таким образом, остается запустить приложение, чтобы убедиться, что связи работают. Если все сделано правильно, то при выборе записи в таблице с клиентами, в таблице счетов будут отображаться только те данные, которые относятся к этому клиенту. Пример можно найти в каталоге Demo\Part4\OneToAny, а с внешним видом этого приложения мы уже знакомы (см. рис. 17.2).

**Тема 7: Компоненты синхронного просмотра**

Специально для отображения связанной информации в БД, имеются 2 компонента, предназначенных именно для этих целей. Это компоненты DBLookupComboBox и DBLookupListBox. Оба они, хотя визуально и похожи на комбинированный и обычный списки, на самом деле, не являются потомками ни стандартных, ни БД-ориентированных компонентов, а происходят от общего для них класса TDBLookupControl, инкапсулирующего как список значений для просмотра, так и его механизм.

Соответственно, свойства этого класса наследуются обоими компонентами синхронного просмотра - как DBLookupComboBox, так и DBLookupListBox. Все они приведены в таблице 19.3.

|  |
| --- |
| Таблица 19.3. Общие свойства DBLookupComboBox и DBLookupListBox |
| **Свойство** | **Тип** | **Описание** |
| DataField | String | Определяет поле, которое будет отображаться данным компонентом |
| DataSource | TDataSource | Определяет источник данных для отображения в списке |
| Field | TField | Указывает на объект типа TField, который представляет собой данный компонент |
| KeyField | String | Определяет ключевое поле таблицы синхронного просмотра (ListSource), которое должно совпадать со значением поля, указанного в DataField |
| KeyValue | Variant | Представляет собой текущее значение ключевого поля |
| ListField | String | Определяет поле или список полей синхронного просмотра в таблице синхронного просмотра |
| ListFieldIndex | Integer | Определяет номер основного поля синхронного просмотра для случая, если в ListField указан список полей |
| ListSource | TDataSource | Определяет компонент источника данных (DataSource), связанный с таблицей синхронного просмотра |
| NullValueKey | TShortCut | Определяет сочетание горячих клавиш для сброса значения на неопределенное |
| ReadOnly | Boolean | Определяет, может ли пользователь изменять значения |

Фактически, здесь следует запомнить лишь следующее: то, что отображается в самом компоненте синхронного просмотра (в обычном или в ниспадающем списке), задается парой значений для ListSource и ListField, а то, на основании чего происходит выборка текущего значения - в DataSource и DataField. При этом для связывания значений используется KeyField. Рассмотрим это на примере наших таблиц счетов и клиентов.

В предыдущем случае мы уже связывали таблицы средствами компонента Table. При этом первичным источником данных ("мастером") являлась таблица клиентов, выбирая запись в которой, мы могли видеть соответствующие ей значения в подчиненной таблице счетов. Но теперь рассмотрим такой вариант, когда первичной информацией являются сами счета. В таком случае, в общем-то, достаточно одной таблицы со счетами. Единственная проблема состоит в том, что в ней виден лишь номер клиента, что удобно машине, но не годится для человека. Соответственно, название клиента можно отображать, воспользовавшись как раз компонентом DBLookupComboBox.

Прежде всего, нам понадобится привычный уже набор из 5 компонент данных, а именно Database и 2 пары Table и DataSource. Первую пару (Table1 и DataSource1) в данном случае мы настроим на таблицу счетов, а вторую - на клиентов. При этом никаких значений для MasterSource и MasterFields нам в данном случае устанавливать не потребуется. После этого пометим на форму таблицу DBGrid и свяжем ее с таблицей счетов, установив свойство DataSource в значение DataSource1. Если база данных подключена, а таблицы активны, то в таблице сразу же будет отображена вся информация о счетах.

Теперь перейдем к собственно синхронному просмотру. Для этого поместим на форму компонент DBLookupComboBox и установим для его свойств следующие значения: DataSource - DataSource1, DataField - BILL\_CUST, ListSource - DataSource2, ListField - CUST\_NAME, и, наконец, свойство KeyFiled установим в CUST\_ID.

Остается лишь запустить приложение и убедиться, что при выборе той или иной записи в таблице счетов, в комбинированном списке синхронного просмотра отображается название соответствующего клиента (рис. 19.7).



Рис. 7. Синхронный просмотр таблиц БД

Если же раскрыть сам список, то в нем можно будет увидеть весь список клиентов, а если выбрать из него другое значение, то цифра в колонке BILL\_CUST так же поменяется. Таким образом, связь получается полноценная, в обе стороны.

Помимо рассмотренных свойств, отвечающих за организацию связи между данными, у компонента DBLookupComboBox имеются и собственные свойства, относящиеся к его визуальной части. Это DropDownAlign, DropDownRows и DropDownWidth, которые отвечают, соответственно, за выравнивание элементов в раскрывающемся списке, за их количество в нем и за ширину окна списка. Кроме них, для чтения во время выполнения программы доступны еще 2 свойства - ListVisible, указывающее на то, раскрыт ли список в данный момент, и Text, содержащее текущее значение списка в виде текстовой строки.

Что касается компонента DBLookupListBox, то с точки зрения организации синхронного просмотра он полностью аналогичен компоненту DBLookupComboBox. Различия касаются лишь визуальной формы представления, а так же собственных свойств, коих у синхронного списка всего 3 - BorderStyle, RowCount и SelecteItem. Первое отвечает за наличие обрамляющей рамки у списка, второе - указывает на количество видимых в списке рядов, а третье аналогично свойству Text у комбинированного списка, т.е. содержит выбранное значение в виде строки.

### Тема 8: Модуль хранения компонентов данных

До настоящего момента мы рассматривали лишь простейшие случаи, с использованием 1-2 таблиц и такого же небольшого числа источников данных. Если же говорить о реальных приложениях БД, то число невизуальных компонентов, используемых в программе для доступа к данным, нередко исчисляется десятками. Такое их изобилие грозит превратить форму главного окна в одно сплошное нагромождение компонент, изрядно мешая работе. Кроме того, одни и те же компоненты могут понадобиться в разных окнах приложения, в то время, как включать в список используемых модулей главное (или любое другое) окно только лишь для ссылки на данные не представляется идеальным вариантом. Поэтому в Delphi предусмотрено специальная оконная форма - DataModule, предназначенная исключительно для размещения на ней невизуальных компонент для доступа к данным.

Отличие окна DataModule от обычной формы состоит в том, что на нем можно размещать только невизуальные компоненты. Это могут быть не только компоненты для доступа к данным, но, в принципе, и любые другие, необходимые в разных частях приложения.

На таком окне можно совершенно свободно, без лишней скученности и без оглядок на пользовательский интерфейс, расположить весьма внушительное число необходимых компонентов (рис. 19.8).



Рис. 8. DataModule может вместить в себя множество компонентов доступа к данным

Для создания окна DataModule следует из меню File > New>OTHER выбрать пункт Data Module. После чего достаточно будет назначить имя этому модулю (например, DM) и сохранить файл, назвав его, скажем, data.pas. После этого, как и в случае с обычными формами, можно будет его включать в конструкцию uses. После этого становится возможным ссылаться на источники данных через стандартную точечную нотацию, используя имя модуля данных:

DBGrid1.DataSource:=DM.DataSource;

**Темы индивидуальных заданий (примеры)**

|  |
| --- |
| 1. Организация деятельности администрации гостиницы.
 |
| 1. Организация работы службы автоинспекции.
 |
| 1. Деятельность налоговой службы.
 |
| 1. Организация работы службы социальной помощи.
 |
| 1. Деятельность абонентской службы АТС.
 |
| 1. Организация работы рекламного агентства.
 |
| 1. Деятельность службы трудоустройства.
 |
| 1. Организация работы службы общественного питания.
 |
| 1. Организация работы службы скорой помощи.
 |
| 1. Деятельность фирмы бартерного обмена.
 |

**ПРИМЕР РАЗРАБОТКИ ПРИЛОЖЕНИЯ**

**Технология доступа к данным средствами Delphi**

## Содержание

**Введение** 3

**1 Теоретическая часть**

1.1 MS Access 4

1.2 Технология ADO 5

1.3 Технология SQL 6

**2 Практическая часть**

2.1 Описание предметной области 8

2.2 Датологическое проектирование 8

2.3 Проектирование БД «Библиотека» в среде СУБД Access 9

2.4 Функциональность программы 10

2.5 Формализация задачи 10

2.6 Схема решения задачи 11

**3 Логика работы программы**

3.1 Архитектура программно-аппаратного комплекса 12

3.2 Руководство пользователя 12

**Заключение** 19

**Список использованной литературы** 20

**Приложение: листинг программы** 21

# Введение

Основные идеи современной информационной технологии базируются на концепции, согласно которой данные должны быть организованы в базы данных с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Эти базы данных создаются и функционируют под управлением специальных программных комплексов, называемых системами управления базами данных (СУБД). [1]

Данная работа посвящена упрощению одной из сторон практической деятельности с использованием баз данных.

База данных – это поименованная совокупность структурированных данных предметной области.

Учитывая интегрирование компьютерных технологий в повседневную жизнь, такие учреждения как, например, библиотеки могут использовать компьютерные программы для хранения различных данных (информация о книгах, об абонементах, о читателях и т.д.).

Целью курсовой работы является создание программы «BDLibrary», которое упрощает работу сотрудников библиотеки, что приводит к экономии большого количества времени.

Задачами курсовой работы является систематизация, закрепление и расширение теоретических и практических знаний по использованию готовых и созданию собственных баз данных, применяя систему объектно-ориентированного программирования Delphi, а также развитие навыков ведения самостоятельной работы.

Предметом исследования является библиотека.

# 1 Теоретическая часть

**1.1 MS Access**

Приложение Microsoft Access 97/2000 (далее Access) является мощной и высокопроизводительной 32-разрядной системой управления реляционной базой данных (далее СУБД).

**База данных** – это совокупность структурированных и взаимосвязанных данных и методов, обеспечивающих добавление выборку и отображение данных.

Реляционная база данных. Практически все СУБД позволяют добавлять новые данные в таблицы. С этой точки зрения СУБД не отличаются от программ электронных таблиц (Excel) ,которые могут эмулировать некоторые функции баз данных. Существует три принципиальных отличия между СУБД и программами электронных таблиц:

* СУБД разрабатываются с целью обеспечения эффективной обработки больших объёмов информации, больших, чем те, с которыми справляются электронные таблицы.
* СУБД может легко связывать две таблицы так, что для пользователя они будут представляться одной таблицей. Реализовать такую возможность в электронных таблицах практически невозможно.
* СУБД минимизируют общий объём базы данных. Для этого таблицы, содержащие повторяющиеся данные, разбиваются на несколько связанных таблиц.

Access – мощное приложение Windows. При этом производительность СУБД органично сочетаются со всеми удобствами и преимуществами Windows. [3]

Как реляционная СУБД Access обеспечивает доступ ко всем типам данных и позволяет одновременно использовать несколько таблиц базы данных. Можно использовать таблицы, созданные в среде Paradox или dBase. Работая в среде Microsoft Office , пользователь получает в своё распоряжение полностью совместимые с Access

текстовые документы(Word) , электронные таблицы(Excel) , презентации(PowerPoint).С помощью новых расширений для Internet можно напрямую взаимодействовать с данными из World Wide Web и транслировать представление данных на языке HTML, обеспечивая работу с такими приложениями как Internet Explorer и Netscape Navigator.

Access специально спроектирован для создания многопользовательских приложений , где файлы базы данных являются разделяемыми ресурсами в сети. В Access реализована надёжная система защиты от несанкционированного доступа к файлам.

База данных храниться в одном файле, но профессиональные пользователи предпочитают разделять базу данных на два файла: в одном хранятся объекты данных (таблицы, запросы), в другом объекты приложения (формы, отчёты, макросы, модули).

Microsoft Access — реляционная СУБД корпорации Microsoft. Имеет широкий спектр функций, включая связанные запросы, сортировку по разным полям, связь с внешними таблицами и базами данных. Благодаря встроенному языку VBA, в самом Access можно писать приложения, работающие с базами данных.

База данных. В общем смысле - совокупность сведений о конкретных объектах реального мира в какой-либо предметной области или разделе предметной области. В терминах СУБД MS Access база данных - это набор данных и объектов, связанных общей задачей. Каждая база данных - это совокупность таблиц, запросов, форм, отчётов, макросов и модулей, которая хранится в файле с произвольным именем и расширением .MDB.

Microsoft Access объединяет сведения из разных источников в одной реляционной базе данных. Создаваемые формы, запросы и отчеты позволяют быстро и эффективно обновлять данные, получать ответы на вопросы, осуществлять поиск нужных данных, анализировать данные, печатать отчеты.

**1.2 Технология ADO**

Наряду с традиционными инструментами доступа к данным Borland Database Engine и ODBC в приложениях Delphi можно применять технологию Microsoft ActiveX Data Objects (ADO), которая основана на возможностях СОМ, а именно интерфейсов OLE DB.

Технология ADO завоевала популярность у разработчиков, благодаря универсальности — базовый' набор интерфейсов OLE DB имеется в каждой современной операционной системе Microsoft. Поэтому для обеспечения доступа приложения к данным достаточно лишь правильно указать провайдер соединения ADO и затем переносить программу на любой компьютер, где имеется требуемая база данных и, конечно, установленная ADO. [4]

Провайдеры ADO обеспечивают соединение приложения, использующего данные через ADO, с источником данных (сервером SQL, локальной СУБД, файловой системой и т. д.). Для каждого типа хранилища данных должен существовать провайдер ADO.

Провайдер "знает" о местоположении хранилища данных и его содержании, умеет обращаться к данным с запросами и интерпретировать возвращаемую служебную информацию и результаты запросов с целью их передачи приложению.

Список установленных в данной операционной системе провайдеров доступен для выбора при установке соединения через компонент TADOConnection.

При инсталляции Microsoft ActiveX Data Objects в операционной системе устанавливаются следующие стандартные провайдеры.

* Microsoft Jet OLE DB Provider обеспечивает соединение с данными СУБД Access при посредстве технологии ОАО.
* Microsoft OLE DB Provider for Microsoft Indexing Service обеспечивает доступ только для чтения к файлам и Internet-ресурсам Microsoft Indexing Service.
* Microsoft OLE DB Provider for Microsoft Active Directory Service обеспечивает доступ к ресурсам службы каталогов (Active Directory Service).
* Microsoft OLE DB Provider for Internet Publishing позволяет использовать ресурсы, предоставляемые Microsoft FrontPage, Microsoft Internet Information Server, HTTP-файлы.
* Microsoft Data Shaping Service for OLE DB позволяет использовать иерархические наборы данных.
* Microsoft OLE DB Simple Provider предназначен для организации доступа к источникам данных, поддерживающим только базисные возможности OLE DB.
* Microsoft OLE DB Provider for ODBC drivers обеспечивает доступ к данным, которые уже "прописаны" при помощи драйверов ODBC. Однако реальное использование столь экзотичных вариантов соединений представляется проблематичным. Драйверы ODBC и так славятся своей медлительностью, поэтому дополнительный слой сервисов здесь ни к чему.
* Microsoft OLE DB Provider for Oracle обеспечивает соединение с сервером Oracle.
* Microsoft OLE DB Provider for SQL Server обеспечивает соединение с сервером Microsoft SQL Server.

В программе «BDLibrary» используется провайдер Microsoft Jet OLE DB Provider, так как таблицы данных, с которыми она взаимодействует, были созданы в СУБД Access. [5]

**1.3 Технология SQL**

Компоненты Delphi для работы с базами данных были созданы в расчете на работу с SQL и архитектурой клиент/сервер. При работе с ними вы можете воспользоваться характеристиками расширенной поддержки удаленных серверов. Delphi осуществляет эту поддержку двумя способами. Во-первых, непосредственные команды из Delphi позволяют разработчику управлять таблицами, устанавливать пределы, удалять, вставлять и редактировать существующие записи. Второй способ заключается в использовании запросов на языке SQL, где строка запроса передается на сервер для ее разбора, оптимизации, выполнения и передачи обратно результатов. [2]

Delphi поддерживает статический и динамический SQL. В Delphi имеется объект TQuery, который используется для хранения и выполнения SQL-запросов.

Свойство TQuery SQL содержит текст SQL-запроса, выполняемых TQuery.
Данное свойство имеет тип TStrings, означающее, что оно может хранить в списке целую серию строк. Список ведет себя подобно массиву, но в действительности это специальный класс с уникальными возможностями.

**Компонент TQuery позволяет выполнять два типа SQL-запросов:**

* Статические SQL-запросы
* Динамические SQL-запросы

Статический SQL-запрос устанавливается во время проектирования и не содержит никаких параметров или переменных. Например, следующая строка является статическим SQL-запросом:

SELECT \* FROM CUSTOMER WHERE CUST\_NO = 1234

Динамический SQL-запрос, или, как его еще называют, параметрический запрос, включает в себя параметры для колонок или имени таблицы. Например, следующая строка является динамическим SQL-запросом:

SELECT \* FROM CUSTOMER WHERE CUST\_NO = :Number

Переменная Number, указанная после двоеточия - параметр, который вы назначаете во время выполнения приложения. Во время выполнения запроса параметр может изменяться.

**Delphi-приложения могут использовать SQL для получения доступа к следующим БД:**

* Таблицы Paradox или dBASE, использующие локальный SQL. Допустимый синтаксис является подмножеством стандарта ANSI-standard SQL и включает основные SELECT, INSERT, UPDATE, и DELETE запросы. Для получения дополнительной информации о локальном синтаксисе SQL обратитесь к справке Using Local SQL.
* Базы данных Local InterBase Server, включая Local InterBase Server. Допускаются любые запросы InterBase SQL. Для получения дополнительной информации о синтаксисе и ограничениях обратитесь к электронной справке SQL Statement and Function Reference.
* Базы данных на удаленных серверах баз данных (только в версии Delphi Client/server). По-видимому вы установили подходящий SQL Link. В SQL серверах допускаются любые стандартные SQL запросы. Для получения дополнительной информации о синтаксисе и ограничениях обратитесь к электронной справке вашего сервера. [6]

# 2 Практическая часть

**2.1 Описание предметной области**

Необходимо построить базу данных, содержащую информацию о выдаче книг читателям в библиотеке. Такая база данных должна обеспечить хранение и просмотр данных о выдаче книг.

 Информационное обеспечение такого приложения пользователя включает:

* Информацию о книгах.
* Информацию о читателях.
* Информацию об абонементах.

 **2.2 Даталогическое проектирование**

 Логическая структура реляционной базы данных Access является адекватным отображением полученной информационно-логической модели предметной области. Для канонической модели не требуется дополнительных преобразований. Каждый информационный объект модели данных отображается реляционной таблицей. Структура реляционной таблицы определяется реквизитным составом соответствующего информационного объекта, где каждое поле соответствует одному из реквизитов объекта. Ключевые реквизиты объекта образуют уникальный ключ реляционной таблицы. Для каждого поля задается тип, размер данных и др. свойства. Записи таблицы соответствуют экземплярам объекта и формируются при загрузке таблицы.

 Связи между объектами данных реализуются одинаковыми реквизитами – ключами связи в соответствующих таблицах. При этом ключом связи типа 1:М всегда является уникальный ключ главной таблицы.



Рис.1 Схема базы данных

**2.3 Проектирование БД «Библиотека» в среде СУБД Access**

1. Структура таблицы «Книги» следующая:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип | Свойство | Описание |
| Код\_книги | Текстовый | Размер 5 | Библиотечный шифр |
| Название | Текстовый | Размер 20 | Название книги |
| Автор | Текстовый | Размер 20 | Автор книги |
| Жанр | Текстовый | Размер 20 | Жанр книги |
| Год\_выпуска | Числовой |  | Год выпуска книги |
| Цена\_за\_сутки | Числовой |  | Суточная цена книги |

2. Структура таблицы «Читатели» следующая:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип | Свойство | Описание |
| Код\_абонента | Текстовый | Размер 5 | Шифр абонента |
| Фамилия | Текстовый | Размер 20 | Фамилия абонента |
| Адрес | Текстовый | Размер 20 | Адрес абонента |
| Телефон | Текстовый | ##-##-## | Телефон абонента |

 3. Структура таблицы «Абонемент» следующая:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип | Свойство | Описание |
| Код\_книги | Текстовый | Размер 5 | Библиотечный шифр |
| Код\_абонента | Текстовый | Размер 5 | Шифр абонента |
| Дата\_выдачи | Дата/Время | ##.##.## | Дата выдачи книги |
| Дата\_возврата | Дата/Время | ##.##.## | Дата возврата книги |

# 2.4 Функциональность программы

Данное приложение должно осуществлять просмотр и редактирование таблиц баз данных, фильтровать данные, выполнять сортировку, выполнять поиск и отправлять отчеты на печать.

 Задача, решаемая с помощью разрабатываемого программного обеспечения, заключается в следующем:

* Предусмотреть удобный ввод данных;
* Отображение и просмотр данных «Книги», «Читатели» и «Абонемент» в табличном виде;
* Провести сортировку данных таблицы по выбранным полям; произвести фильтрацию данных, выбираемых из раскрывающегося списка; выполнить фильтрацию данных и поиск абонента или книги по коду; подсчитать общее количество книг, взятых каждым абонентом; выполнить поиск по фамилии абонента;
* Просмотр диаграммы;
* Просмотр отчетов;
* Содержать справочную информацию об авторе и о программе;

**2.5 Формализация задачи**

При работе приложения используются следующие промежуточные переменные:

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение | Обозначение (имя) | Тип | Ограничения |
| Вспомогательная переменная для выполнение запросов | s | String | #0..#255 |
| Вспомогательная переменная для выполнение запросов | p | String | #0..#255 |
| Вспомогательная переменная для выполнение поиска | f | boolean |  |
| Вспомогательная переменная для выполнение поиска | c | variant |  |

В базе данных находятся следующие таблицы:

Таблица «Книги»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Ключевое поле |
| Код\_книги | Текстовый | Размер 5 | \* |
| Название | Текстовый | Размер 20 |  |
| Автор | Текстовый | Размер 20 |  |
| Жанр | Текстовый | Размер 20 |  |
| Год\_выпуска | Числовой |  |  |
| Цена\_за\_сутки | Текстовый |  |  |

Таблица «Читатели»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Ключевое поле |
| Код\_абонента | Текстовый | 5 | \* |
| Фамилия | Текстовый | 20 |  |
| Адрес | Текстовый | 20 |  |
| Телефон | Текстовый | 8 |  |

Таблица «Абонемент»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Ключевое поле |
| Код\_книги | Текстовый | 5 |  |
| Код\_абонента | Текстовый | 5 |  |
| Дата\_выдачи | Дата/время | Короткий формат даты |  |
| Дата\_возврата | Дата/время | Короткий формат даты |  |

**2.6 Схема решения задачи**

# 3 Логика работы программы

Главная форма

Данные

Работа с данными

О программе

Просмотр редактирование

Выход

Группировка

Поиск по коду

Поиск по фамилии

Сортировка

**3.1 Архитектура программно-аппаратного комплекса**

Для корректной работы приложения необходимы следующие аппаратные и системные программные средства:

**ADO**

 Использование ADO является альтернативой Borland Database Engine (BDE), обеспечивающей более эффективную работу с данными. Для использования этой возможности на вашем компьютере должна быть установлена система ADO 2.1 или более старшая версия. Кроме того должна быть установлена клиентская система доступа к данным, например, Microsoft SQL Server, а в ODBC должен иметься драйвер OLE DB для того типа баз данных, с которыми вы работаете.

**Microsoft Office Access 2003**

Microsoft Office Access 2003 является системой управления базами данных в которой можно создавать и редактировать базы данных.

Microsoft Office Access 2003 позволяет просматривать сведения о зависимостях между объектами базы данных. Просмотр списка объектов, используемых указанным объектом, помогает осуществлять поддержку базы данных и предотвращать ошибки, связанные с потерей источников записей.

**Системные требования**

Процессор Intel Pentium(R) 4 Celeron 2.40 GHz, 768 MB ОЗУ, RADION 9250 AGP 128MB, 10MB свободного места на жестком диске, ОС Microsoft Windows XP Pro SP2.

**3.2 Руководство пользователя**

При открытии приложения открывается главное окно программы (рис. 2)



Рис.2 Главное окно программы

Меню программы содержит следующие пункты:

* **Данные**

С помощью пункта меню «Данные» вы можете отредактировать имеющиеся в базе данных таблицы. Кроме редактирования таблиц вы можете также их просто просмотреть Для этого нужно выполнить следующее: Данные – Просмотр и редактирование (рис. 3), после чего откроется окно заполнения таблиц (рис. 4).



Рис.3 Меню «Ввод данных»



Рис.4 Форма просмотра и редактирования данных

* **Работа с данными**

С помощью пункта меню «Работа с данными» (рис. 5) вы сможете отсортировать записи по выбранным вами полям: по возрастанию и убыванию, просмотреть различные виды поисков и подсчитать общее количество книг у каждого абонента. Для этого нужно выполнить следующее: Работа с данными – Поиск по коду (рис. 6), Работа с данными – Группировка (рис.7), Работа с данными – Поиск по Фамилии (рис.8), Работа с данными – Сортировка (рис.9).



Рис.5 Меню «Работа с данными»

 



Рис.6 Окно приложения «Поиск по коду». Код поиска



 

Рис.7 Окно приложения «Группировка». Код группировки





Рис.8 Окно приложения «поиск по фамилии». Код поиска



 

Рис.9 Окно приложения «Сортировка». Код сортировки

* **Справка**

С помощью пункта меню «Справка» вы можете получить информацию об авторе и о программе.



Рис.10 Окно приложения «О программе»

* **Выход**

После выбора пункта Меню “Выход” программа будет закрыта.

**Заключение**

В ходе выполнения курсовой работы была разработана программа «BDLibrary», обладающая следующим функционалом:

1) просмотр и редактирование данных;

2) поиск по коду;

3) группировка данных;

4) поиск по фамилии;

5) сортировка данных;

6) справка о программе

В заключении можно сказать, что созданная программа – проста, удобна в применении, так как данные отображаются в таблице, экономит большое количество времени. Эта программа доступна для любого человека, работающего в библиотеке и обладающего простыми навыками пользования компьютером.

Потенциал программы большой. В будущем, функциональность программы можно было бы расширить за счет добавления различных функций. Например, добавить в программу несколько новых видов поиска, сортировки, добавить функцию просмотра данных в виде диаграмм (для большей наглядности данных). Также можно добавить возможность создавать по данным таблиц отчеты, которые можно было бы распечатывать.

# Список использованной литературы

1. А.Н. Морган «Справочник по Turbo Pascal для студентов», Диалектика, 2006
2. А.Я. Архангельский «Программирование в Delphi 2005 и 2006», Издательство «БИНОМ», 2006
3. Боровский, А.Н. Программирование в Delphi/ А.Н. Боровский. - СПб.: БХВ-Петербург. – 2005.
4. Симонович, С.В. Информатика. Базовый курс./ С.В.Симонович/ 2-е издание, под ред. С. В. Симоновича. - СПб.: Питер. - 2005.
5. <http://www.realcoding.net>
6. <http://www.delphisources.ru>

**Приложение: листинг программы**

**Unit1.pas**

unit Unit1;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, Menus, XPMan, ExtCtrls;

type

 TForm1 = class(TForm)

 MainMenu1: TMainMenu;

 N1: TMenuItem;

 N2: TMenuItem;

 N3: TMenuItem;

 N6: TMenuItem;

 N7: TMenuItem;

 N8: TMenuItem;

 N4: TMenuItem;

 Image1: TImage;

 N5: TMenuItem;

 N9: TMenuItem;

 procedure N2Click(Sender: TObject);

 procedure N6Click(Sender: TObject);

 procedure N8Click(Sender: TObject);

 procedure N7Click(Sender: TObject);

 procedure N4Click(Sender: TObject);

 procedure N5Click(Sender: TObject);

 procedure N9Click(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form1: TForm1;

implementation

uses Unit2, Unit3, Unit4, Unit5, Unit6, Unit7;

{$R \*.dfm}

procedure TForm1.N2Click(Sender: TObject);

begin

form2.show;

form1.Hide;

end;

procedure TForm1.N6Click(Sender: TObject);

begin

form3.show;

form1.Hide;

end;

procedure TForm1.N8Click(Sender: TObject);

begin

form4.show;

form1.Hide;

end;

procedure TForm1.N7Click(Sender: TObject);

begin

form5.show;

form1.Hide;

end;

procedure TForm1.N4Click(Sender: TObject);

begin

form6.show;

form1.hide;

end;

procedure TForm1.N5Click(Sender: TObject);

begin

Form1.Close;

end;

procedure TForm1.N9Click(Sender: TObject);

begin

form7.show;

form1.Hide;

end;

end.

**Unit2.pas**

unit Unit2;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, ExtCtrls, DBCtrls, Grids, DBGrids, ComCtrls, DB, ADODB, XPMan;

type

 TForm2 = class(TForm)

 ADOConnection1: TADOConnection;

 ADOTable1: TADOTable;

 DataSource1: TDataSource;

 PageControl1: TPageControl;

 TabSheet1: TTabSheet;

 TabSheet2: TTabSheet;

 TabSheet3: TTabSheet;

 DBGrid1: TDBGrid;

 DBNavigator1: TDBNavigator;

 ADOConnection2: TADOConnection;

 ADOTable2: TADOTable;

 DataSource2: TDataSource;

 DBGrid2: TDBGrid;

 DBNavigator2: TDBNavigator;

 ADOConnection3: TADOConnection;

 ADOTable3: TADOTable;

 DataSource3: TDataSource;

 DBGrid3: TDBGrid;

 DBNavigator3: TDBNavigator;

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form2: TForm2;

implementation

{$R \*.dfm}

end.

**Unit3.pas**

unit Unit3;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, StdCtrls, Grids, DBGrids, DB, ADODB, ExtCtrls;

type

 TForm3 = class(TForm)

 ADOConnection1: TADOConnection;

 ADOTable1: TADOTable;

 DataSource1: TDataSource;

 DBGrid1: TDBGrid;

 GroupBox1: TGroupBox;

 Label1: TLabel;

 Edit1: TEdit;

 Button1: TButton;

 Button2: TButton;

 RadioGroup1: TRadioGroup;

 RadioButton1: TRadioButton;

 RadioButton2: TRadioButton;

 ADOQuery1: TADOQuery;

 procedure Button1Click(Sender: TObject);

 procedure Button2Click(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form3: TForm3;

implementation

{$R \*.dfm}

procedure TForm3.Button1Click(Sender: TObject);

var p,s:string;

begin

if radiobutton1.Checked=True then p:='Код\_книги';

if radiobutton2.Checked=True then p:='Код\_абонента';

adoquery1.Close;

Adoquery1.sql.Clear;

s:='select \* from абонемент where '+p+'="'+edit1.Text+'"';

Adoquery1.sql.Add(s);

Adoquery1.open;

datasource1.DataSet:=adoquery1;

end;

procedure TForm3.Button2Click(Sender: TObject);

begin

datasource1.DataSet:=adotable1;

end;

end.

**Unit4.pas**

unit Unit4;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, Grids, DBGrids, DB, ADODB, StdCtrls, ComCtrls;

type

 TForm4 = class(TForm)

 ADOConnection1: TADOConnection;

 ADOTable1: TADOTable;

 DataSource1: TDataSource;

 DBGrid1: TDBGrid;

 PageControl1: TPageControl;

 TabSheet1: TTabSheet;

 Label1: TLabel;

 GroupBox1: TGroupBox;

 Label2: TLabel;

 Edit1: TEdit;

 Button1: TButton;

 TabSheet2: TTabSheet;

 Label3: TLabel;

 GroupBox2: TGroupBox;

 Label4: TLabel;

 Edit2: TEdit;

 Button2: TButton;

 Label5: TLabel;

 Label6: TLabel;

 procedure Button1Click(Sender: TObject);

 procedure Button2Click(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form4: TForm4;

implementation

{$R \*.dfm}

procedure TForm4.Button1Click(Sender: TObject);

var f:boolean;

begin

f:=adotable1.Locate('Фамилия',edit1.text,[lopartialkey]);

if f=false then showmessage ('Такой фамилии нет!!!');

end;

procedure TForm4.Button2Click(Sender: TObject);

var c:variant;

begin

c:=adotable1.lookup('Фамилия',edit2.Text,'Телефон');

case vartype(c) of

varempty:label6.Caption:='пустой результат';

varnull:label6.caption:='запись не найдена';

else

label6.Caption:=c;

end;

end;

end.

**Unit5.pas**

unit Unit5;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, ADODB, StdCtrls, Grids, DBGrids, DB;

type

 TForm5 = class(TForm)

 ADOConnection1: TADOConnection;

 ADOTable1: TADOTable;

 DataSource1: TDataSource;

 DBGrid1: TDBGrid;

 Button1: TButton;

 ADOQuery1: TADOQuery;

 Button2: TButton;

 procedure Button1Click(Sender: TObject);

 procedure Button2Click(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form5: TForm5;

implementation

{$R \*.dfm}

procedure TForm5.Button1Click(Sender: TObject);

var s:string;

begin

AdoQuery1.Close;

AdoQuery1.SQL.Clear;

s:='select Читатели.Фамилия, count(Абонемент.Код\_книги) as Количество\_книг from Книги,Читатели,Абонемент where Читатели.Код\_абонента=Абонемент.Код\_абонента and Книги.Код\_книги=Абонемент.Код\_книги group by Читатели.Фамилия';

AdoQuery1.SQL.Add(s);

AdoQuery1.Open;

datasource1.DataSet:=adoquery1;

end;

procedure TForm5.Button2Click(Sender: TObject);

begin

datasource1.DataSet:=adotable1;

end;

end.

**Unit6.pas**

unit Unit6;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, StdCtrls, jpeg, ExtCtrls;

type

 TForm6 = class(TForm)

 Image1: TImage;

 Label1: TLabel;

 Label2: TLabel;

 Label3: TLabel;

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form6: TForm6;

implementation

{$R \*.dfm}

end.

**Unit7.pas**

unit Unit7;

interface

uses

 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

 Dialogs, StdCtrls, ExtCtrls, Grids, DBGrids, DB, ADODB;

type

 TForm7 = class(TForm)

 ADOConnection1: TADOConnection;

 ADOTable1: TADOTable;

 DataSource1: TDataSource;

 DBGrid1: TDBGrid;

 GroupBox1: TGroupBox;

 RadioGroup1: TRadioGroup;

 Label1: TLabel;

 ComboBox1: TComboBox;

 Button1: TButton;

 Button2: TButton;

 RadioButton1: TRadioButton;

 RadioButton2: TRadioButton;

 ADOQuery1: TADOQuery;

 procedure Button1Click(Sender: TObject);

 procedure Button2Click(Sender: TObject);

 private

 { Private declarations }

 public

 { Public declarations }

 end;

var

 Form7: TForm7;

implementation

{$R \*.dfm}

procedure TForm7.Button1Click(Sender: TObject);

var s,p:string;

begin

if ComBobox1.text='' then showmessage('Не выбрано поле сортировки!');

if ComBobox1.text='Код\_книги' then p:='Код\_книги';

if ComBobox1.text='Название' then p:='Название';

if ComBobox1.text='Жанр' then p:='Жанр';

if ComBobox1.text='Год\_выпуска' then p:='Год\_выпуска';

if ComBobox1.text='Цена\_за\_сутки' then p:='Цена\_за\_сутки';

adoquery1.Close;

adoquery1.SQL.Clear;

s:='select\* from Книги order by '+p;

if radiobutton2.Checked=True then s:=s+' desc';

adoquery1.SQL.Add(s);

Adoquery1.open;

datasource1.DataSet:=adoquery1;

end;

procedure TForm7.Button2Click(Sender: TObject);

begin

datasource1.DataSet:=adotable1;

end;

end.

**Используемые источники:**

1. Гагарина Л. Г., Кокорева Е. В., Виснадул Б. Д. Технология разработки программного обеспечения: учебное пособие / под ред. Л. Г Гагариной. — М: ИД «ФОРУМ»: ИНФРА-М, 2008. — 400 с.: ил. — (Высшее образование).
2. Брауде Э. Технология разработки программного обеспечения. — СПб.: Питер, 2004. — 655 с.
3. http://www.studmed.ru.