Талипов С.Н.

Базы данных на Delphi 7

12+

Сергей Николаевич Талипов

Базы данных на Delphi 7

http://www.litres.ru/pages/biblio_book/?art=65506382
SelfPub; 2021

Аннотация

Лекции по базам данных на Delphi 7. Рассмотрены простейшие программы, программы с поиском, фильтрацией, каскадным удалением и транзакцией, а также создание отчетов для печати данных.

Содержание

Сергей Талипов
Базы данных на Delphi 7

ЛЕКЦИЯ № 1

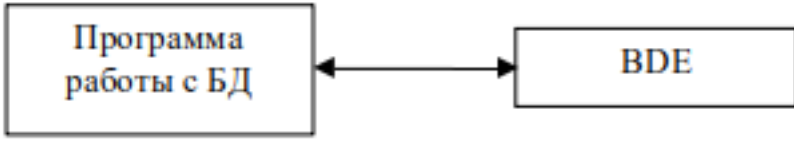
1. СОЗДАНИЕ ПО НА ОСНОВЕ БАЗ ДАННЫХ В DELPHI

Программирование баз данных – очень большой и серьезный раздел самого что ни на есть практического программирования, многие программисты большую часть своего времени тратят именно на проектирование баз данных и разработку приложений, работающих с ними. Это неудивительно – в настоящее время каждая государственная организация, каждая фирма или крупная корпорация имеют рабочие места с компьютерами. Имеется масса данных, которые нужно не только сохранить, но и обработать, получить комплексные отчеты. Без баз данных сегодня не обойтись. А завтра они будут еще нужней.

Недостаточно просто написать программу, взаимодействующую с БД. Нужно уметь правильно спроектировать эту базу данных. Проектирование баз данных, в общем, являет-

ся первым шагом разработки приложения. Только когда база данных спроектирована, программист приступает непосредственно к проекту приложения.

При написании программ работы с базами данных (БД) на «Делфи» обычно используется метод доступа через BDE – систему. BDE – система представляет из себя набор драйверов, связывающих прикладную программу на «Делфи» с физическими файлами БД.



В «Делфи» понятие «таблица данных» и «база данных» различаются. Таблицей называется совокупность данных, нормализованных в табличную форму. Примером таблиц служат классические БД типа FoxPro и DBase. База данных – это файл, содержащий в себе несколько таблиц, которые имеют, как правило, внутренние взаимосвязи по ключевым полям. Базы данных используются в крупных промышленных СУБД типа «ORACLE» и других клиент – серверных системах.

Для ознакомления с программированием баз данных будем пользоваться таблицами типа DBase и FoxPro. Для начала любой работы с БД из под «Делфи» необходимо настро-

ить BDE – систему. Для этого служит специальная программа «BDE Administrator». Для создания таблиц и наполнения их данными можно воспользоваться программой «DataBase Desktop» из установленного комплекта «Делфи».

2. ПРОСТЕЙШАЯ ПРОГРАММА

Для создания простейшей программы понадобится одна форма, не визуальные компоненты «DataSource» и «Table» из вкладки «Data Access», и визуальные компоненты «DBGrid» и «DBGrid» из вкладки «Data Controls». Расположите компоненты на форме как показано на рис.1.

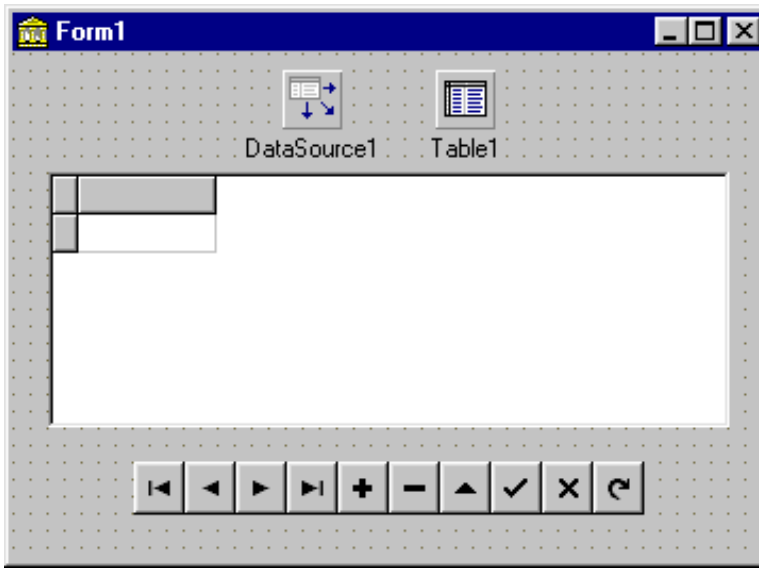


Рис. 1

Не визуальный компонент «Table» предназначен для непосредственного доступа к таблице данных через BDE –

систему. Визуальный компонент «DBGrid» служит для просмотра/редактирования записей в таблице данных, компонент «DBNavigator» служит для перемещения (навигации) по таблице данных. Не визуальный компонент «DataSource» служит посредником между компонентом «Table» и всеми визуальными компонентами.

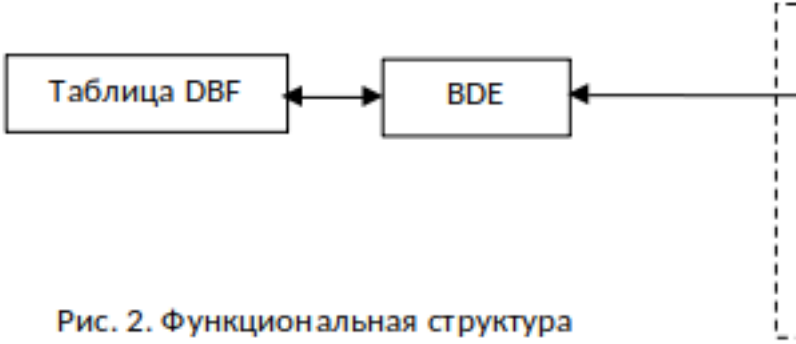


Рис. 2. Функциональная структура простейшей программы работы с БД

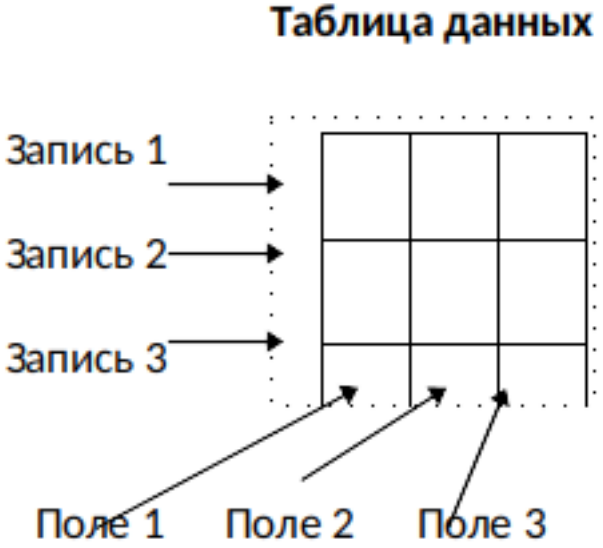


Рис. 3

Программная часть программы состоит из двух обработчиков событий «OnActivate» и «OnClose» для формы. Ниже приведен исходный текст данных обработчиков.

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

```
  begin
  try { Установка защиты на операторы }
    Table1.DatabaseName:='c:\uoo\dbf'; { Установка пути к таблице данных }
   Table1.TableName:='u2_spec.dbf'; { Указание имени файла таблицы данных с расширением }
      DataSource1.DataSet:=Table1; { Связуем компонент Table1 с компонентом DataSource1 }
     DBGrid1.DataSource:=DataSource1; { Связуем компонент DBGrid1 с компонентом DataSource1 }
    DBNavigator1.DataSource:=DataSource1; { Связуем компонент DBNavigator1 с компонентом DataSource1 }
    if not Table1.active then Table1.Open; { Если таблица еще не открыта, то открыть ее }
  except { Если произошла ошибка при открытии таблицы данных, то выдать сообщение }
    showmessage('Error !');
  end;
  end;
  procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
  begin { Если таблица данных открыта, то закрыть ее }
  try if Table1.active then Table1.Close; except end;
  end;
```

ЛЕКЦИЯ № 2

1. МЕТОДЫ И СВОЙСТВА КОМПОНЕНТ ДОСТУПА К ТАБЛИЦАМ ДАННЫХ

Компонент “TTable” служит для непосредственной связи с базой данных (таблицей). Данный компонент имеет ряд основных методов:

Open – открытие базы данных

Close – закрытие базы данных

Refresh (для BDE) / Requery (для ADO) – обновление базы данных с диска

Edit – перевод БД в режим редактирования текущей записи

Post – запоминание изменений для редактируемой текущей записи (вызывается после метода «Edit»)

Cancel – отмена изменений для редактируемой текущей записи (вызывается после метода «Edit», если нужно отменить изменения)

Insert – вставка новой пустой записи в БД

Append – добавление в конец новой пустой записи в БД
Delete – удаление текущей записи из БД
EmptyTable – удаление всех данных в таблице
DeleteTable – удаление БД с диска

First – переход на первую запись БД
Last – переход на последнюю запись БД
Next – переход на следующую запись БД
Prior – переход на предыдущую запись БД
MoveBy (-10) – переход на -10 записей БД (можно на любой целое число до начала или конца БД)

DisableControls – запретить отображение значений из БД во всех связанных с ней визуальных компонентах
EnableControls – разрешить отображение значений из БД во всех связанных с ней визуальных компонентах

Компонент “TTable” имеет ряд основных свойств:
Active – если «True», то база данных открыта
RecNo – выдает номер текущей записи БД
RecordCount – выдает количество записей в БД
FieldValues – доступ к значению поля БД
Modified – если «True», то текущая запись была изменена
Bof – если «True», то текущая запись – первая
Eof – если «True», то текущая запись – последняя

ЛЕКЦИЯ № 3

1. КОМПОНЕНТЫ ОТОБРАЖЕНИЯ ЗАПИСЕЙ БД

Компонент “DBEdit” служит для отображения и изменения значения одного конкретного поля текущей записи базы данных. Основные свойства этого компонента:

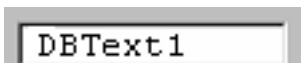
DataField := ‘CODE’; // Имя поля БД для отображения

DataSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД

Компонент “DBText” служит только для отображения значения одного конкретного поля текущей записи базы данных. Основные свойства этого компонента:

DataField := ‘CODE’; // Имя поля БД для отображения

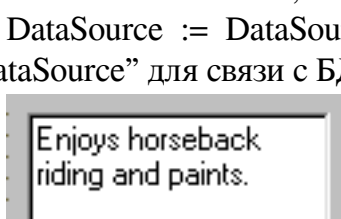
DataSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД



Компонент “DBMemo” служит для отображения значения одного конкретного memo-поля для текущей записи БД. . Основные свойства этого компонента:

DataField := ‘CODE’; // Имя поля БД для отображения

DataSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД



Компонент “DBImage” служит для отображения значения одного конкретного Image-поля для текущей записи БД. Компонент поддерживает только «bmp» формат. Основные свойства компонент:

DataField := ‘CODE’; // Имя поля БД для отображения

DataSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД

Для “DBImage”:

Stretch := true; // Включение режима масштабирования рисунка


```
  procedure TForm1.N1Click(Sender: TObject);
  { Скопировать картинку из таблицы данных в буфер об-
мена }
  begin
    DBImage1.CopyToClipboard;
  end;

  procedure TForm1.N2Click(Sender: TObject);
  { Вставить картинку из буфера обмена в поле таблицы
данных }
  begin
    DBImage1.PasteFromClipboard;
  end;

  procedure TForm1.N3Click(Sender: TObject);
  { Загрузить картинку из файла в таблицу данных }
  begin
```

```
  if opendialog1.Execute=true then begin
    try ADOTable1.edit; except end;
    DBImage1.Picture.LoadFromFile(OpenDialog1.FileName
    try ADOTable1.post; except end;
  end;
end;

procedure TForm1.N4Click(Sender: TObject);
{ Сохранить картинку из таблицы данных в файл }
begin
  if savedialog1.Execute=true then
    DBImage1.Picture.SaveToFile(SaveDialog1.FileName);
end;

procedure TForm1.Button3Click(Sender: TObject);
{ Удаление картинки из базы }
begin
  try ADOTable1.Edit;
  ADOTable1['Oblochka']:=null;
  ADOTable1.Post; except end;
end;
```

Компонент “DBLookupComboBox” служит для выбора конкретной записи БД по значению из конкретного поля базы данных. Данный компонент представляет собой выпадаю-

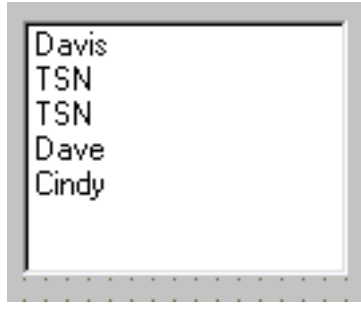
щий список с перечнем значений поля БД для всех записей:

KeyField := ‘CODE’; // Имя поля БД для отображения

ListSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД

Компонент “DBLookupListBox” служит для выбора конкретной записи БД по значению из конкретного поля базы данных. Данный компонент представляет собой прокручиваемый список с перечнем значений поля БД для всех записей:

KeyField := ‘CODE’; // Имя поля БД для отображения

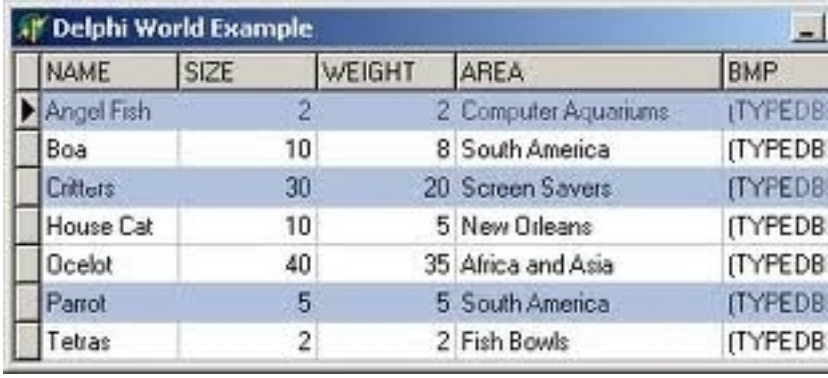


ListSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД

Компонент “DBGrid” служит для отображения значения

всех записей и полей БД. Основные свойства этого компонента:

DataSource := DataSource1; // Имя компонента типа “ DataSource” для связи с БД



2. ПРИМЕР ИСПОЛЬЗОВАНИЯ МЕТОДОВ И СВОЙСТВ КОМПОНЕНТА «TTABLE»

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Table1.Append;
Table1.FieldValues['Name'] := Edit1.text;
Table1. ['Year'] := StrToInt(Edit2.text);
Table1.Post;
end;
```

* * * *

```
Table1.Edit;
Table1.FieldByName('Name').AsString := 'Fred';
Table1.Post;
```

* * * *

```
  Table1.Insert;
  Table1['Name'] := 'Russia';
  Table1['Sity'] := 'Moscow';
  Table1.Post;
```

* * * *

```
if MessageDlg('Сохраннить запись?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then Table1.Post else Table1.Cancel;

* * * *

procedure TForm1.Button1Click(Sender: TObject);
var i: integer; k: real;
begin
  k:=0;
  with ProgressBar1 do begin
    Min := 0; Max := Table1.RecordCount;
    Table1.First;
    for i := Min to Max do begin
      Position := i; s:=s+ Table1['sum'];
      Table1.Next;
    end;
  end;
end;

* * * *
with Table1 do begin
  DisableControls;
  try
    First;
```

```
    while not EOF do Delete;
  finally
    EnableControls;
  end;
end;
```

ЛЕКЦИЯ № 4

1. СОЗДАНИЕ ПОЛЕЙ ВЫБОРА И ВЫЧИСЛЯЕМЫХ ПОЛЕЙ

Поля выбора «Lookup» используются для создания виртуальных полей, данные в которых пользователь не набирает вручную, а выбирает из выпадающего списка. После выбора значения из списка оно отображается на экране, а в файл БД записывается код выбранного значения из выпадающего списка в соответствующее поле БД. Выпадающий список формируется из каких либо двух полей другой таблицы. Одно поле другой таблицы хранит код записи, а другое соответствующий текст, который появляется в выпадающем списке. Lookup-поля используются для подключения к основной таблице вспомогательных таблиц-справочников БД. Рассмотрим применение полей выбора на примере.

Например, имеется следующие таблицы:

Таблица 1. Диски – CD (Table_CD)

Код диска	**Название диска**	**Код фирмы**
	(Name_CD)	**(Kod_Firm)**

0000001	Итнернет – 2001	00001
0000002	Суперсистем-ный диск 2k2	00002
0000003	English Platinum (2)	00003
0000004	All stars disco 2	00004
0000005	Золотая бух-галтерия 2001	00002

Таблица 2. Фирмы – поставщики (Table_Firm)

Код фирмы (Kod_Firm)	**Название фир-мы (Name_Firm)**
00001	Красные Челны
00002	Технопром
00003	CD-маркер
00004	Мегаполис

Данные таблицы имеют общее поле «Код фирмы». Необходимо сделать так, чтобы при просмотре первой таблицы вместо кодов фирм выходило соответствующее название фирмы. Для этого необходимо в первой таблице поле «код фирмы» сделать невидимым для визуальных компонент, и добавить к таблице «Lookup»-поле. Данное поле просмотра

будет смотреть код фирмы в первой таблице у каждой записи, находить соответствующую запись с данным кодом во второй таблице, брать из второй таблицы соответствующее коду название фирмы и подставлять его в качестве своего значения. В результате первая таблица будет отображаться следующим образом:

Таблица 1. Диски – CD (Table_CD)

Код диска	**Название диска**	**Фирма (Loolup-поле)**
	(Name_CD)	**(nFirm)**
0000001	Итнернет – 2001	Красные Челны
0000002	Суперсистемный диск 2k2	Технопром
0000003	English Platinum (2)	CD-маркер
0000004	All stars disco 2	Мегаполис
0000005	Золотая бухгалтерия 2001	Технопром

Если при просмотре таблицы 1 мы поменяем для первой записи значение фирмы «Красные челны» на «Технопром» (через выпадающий список), то в поле «Код фирмы» таблицы 1 запишется значение «00002», взятое из таблицы

2. Таким образом, поля просмотра позволяют хранить в базе данных только нужные коды, а их текстовые значения брать из другой базы данных и подставлять для просмотра и выбора.

Для создания поля выбора необходимо иметь два компонента «Table», один для основной базы, другой – как справочник с расшифровкой поля. Пусть таблица «Table_CD» будет соответствовать таблице 1, а второй компонент «Table_Firm» будет соответствовать таблице 2. Таблица «Table_CD» будет иметь поля «Kod», «Name_CD» и «Kod_firm», а таблица 2 «Table_Firm» будет иметь поля «Kod_firm» и «Name_firm».

Настроим у обоих компонент «Table» путь к БД и имена таблиц. Откроем базу данных «Table_CD», установив свойству «Active» значение «True». После этого щелкнем на компоненте «Table_CD» правой кнопкой мыши. В появившемся меню выберем опцию «Fields Editor». Откроется окно редактора полей. В редакторе полей нажмите правую кнопку мыши и выберите пункт «Add all fields». После этой команды в редакторе полей появится список из имеющихся полей в базе данным, например, поля «Kod», «Name_CD» и «Kod_firm». Эти три поля являются физическими полями, хранящимися в файле базы данных.

Если для компонента «Table» не указаны явно поля через редактор полей, то база данных будет также работоспособна, как и с указанием полей. Отличие заключается лишь в том, что нельзя создавать поля выбора и вычисляемые поля без явного указания всех имеющихся физических полей в редакторе полей.

Создадим дополнительное «Lookup» поле выбора «nFirm» у компонента «Table_CD». Для этого в редакторе полей нужно щелкнуть правой кнопкой мыши и выбрать пункт «New Field». Откроется окно создания нового поля. Зададим параметру «Name» имя создаваемого «lookup» поля, – «nFirm». Тип поля зададим в параметре «Type». Выберем денежный тип «String» с размером «Size» равным 20 символам. После этого укажем в разделе «Field type» значение «Lookup». Это означает, что создаваемое поле будет полем выбора. Далее зададим в параметре «KeyFields» значение «Kod_firm». Это поле таблицы «Table_CD», в котором хранится код фирмы. Далее зададим параметр «Dataset» равным «Table_Firm». Этим параметром указывается имя подчиненной таблицы с расшифровкой кодов фирм. В параметре «Lookup Keys» зададим значение «Kod_firm». Это значение является именем поля таблицы «Table_Firm» с кодами фирм. В параметре «Result Field» укажем имя «Name_firm». Это значение является именем поля таблицы «Table_firm» с названием фирмы. Нажмем теперь на кнопку «Ok». Поле

просмотра создано.

После создания поля просмотра в таблице «Table_CD» поле «Kod_firm» можно сделать невидимым в компонентах просмотра БД, т.к. теперь это поле более наглядно отображает и позволяет изменять значение созданное поле просмотра с именем «nFirm».

В программе Lookup-поля просмотра можно использовать только для чтения. Присваивать значение или изменять их в программе нельзя.

Схема создания Lookup-поля:

Table1 -> Active -> Fields Editor -> Add All Fields -> New Field:

Name: nFirm, Field Type: Lookup, Type: String, Size: 20

Key Fields: 'Kod_Firm' (поле Table1),

DataSet: 'Table_Firm' (ссылка на Table2),

Lookup Keys: 'Kod_Firm' (поле Table2), Result Field: 'Name_Firm'(поле Table2)

2. СОЗДАНИЕ ВЫЧИСЛЯЕМЫХ ПОЛЕЙ У КОМПОНЕНТА «TABLE»

Вычисляемые поля – это виртуальные поля, которые формируются динамически в процессе выполнения программы. В базе данных вычисляемые поля не сохраняются. Вычисляемое поле формирует для каждой записи базы данных значение, получаемое по какой-либо формуле из значений других полей для той же самой записи. Например, имеется БД с полем «Количество товара» и полем «Цена товара». Необходимо определить стоимость каждого товара. Для определения стоимости нужно для каждой записи находить произведение значений из поля «Количество товара» и поля «Цена товара». Введение вычисляемого поля в компонент «Table» позволяет автоматизировать эту задачу: у каждой записи появится новое поле, содержащее готовое произведение цены товара на его количества. Рассмотрим на примере как это сделать.

Нанесем на форму компонент «Table». Настроим у него путь к БД и имя таблицы. Откроем базу данных, установив свойству «Active» значение «True». После этого щелкнем на компоненте «Table» правой кнопкой мыши. В появившемся меню выберем опцию «Fields Editor». Откроется окно редак-

тора полей. В редакторе полей нажмите правую кнопку мыши и выберите пункт «Add all fields». После этой команды в редакторе полей появится список из имеющихся полей в базе данных, скажем, поля «Cena» и «Kolichestvo». Эти два поля являются физическими полями, хранящимися в файле базы данных.

Создадим дополнительное вычисляемое поле «Summ» у компонента «Table». Для этого в редакторе полей нужно щелкнуть правой кнопкой мыши и выбрать пункт «New Field». Откроется окно создания нового поля. Зададим параметру «Name» имя создаваемого вычисляемого поля, – «Summ». Тип поля зададим в параметре «Type». Выберем денежный тип «Currency». После этого укажем в разделе «Field type» значение «Calculated». Это означает, что создаваемое поле будет вычисляемого вида. Нажмем теперь на кнопку «Ok». Вычисляемое поле создано. Данное поле имеет имя «Summ», а полное имя будет «Table1Summ». Полное имя поля можно посмотреть, выбрав в редакторе полей нужное поле и посмотреть его свойство «Name» в инспекторе объектов.

После создания вычисляемого поля необходимо описать алгоритм его формирования. Для этого нужно в Инспекторе Объектов найти событие «OnCalcFields» для компонента «Table» (в котором создано вычисляемое поле) и создать для

данного события обработчик (обработчик создается двойным щелчком по белому пространству, напротив от названия события).

В созданной пустой процедуре-обработчике вычисляемых полей компонента «Table» напишем следующую строку:

Table1Summ . Ascurrency := Table1Cena . Ascurrency * Table1Kolichestvo . Ascurrency;

В обработчике вычисляемых полей можно обращаться к полям только текущей записи, причем поля должны быть записаны в полной форме, т.е. нужно писать не «Summ», а «Table1Summ». После имени поля через точку указывается его тип. Команды перемещения по записям в БД, удаления записей, вставки и др. команды не допустимы в обработчике вычисляемых полей.

В основное программе вычисляемые поля можно использовать только для чтения. Присваивать значение или изменять его в основной программе нельзя. Все операции по корректировки вычисляемых полей должны производится в обработчике «OnCalcFields» компонента «Table».

Если в компоненте «Table» создано несколько вычисляемых полей, то все их алгоритмы формирования описывают-

ся в одном и том же обработчике события «OnCalcFields», например:

```
procedure TForm1.Table1CalcFields(DataSet: TDataSet);
begin
Table1Summ . Ascurrency := Table1Cena . Ascurrency * Table1Kolichestvo . Ascurrency;
Table1Summ2 . Ascurrency := (Table1Cena . Ascurrency * Table1Kolichestvo . Ascurrency) – Table1Nalog . Ascurrency;
end;
```

Схема создания вычисляемого поля:

Table1 -> Active -> Fields Editor -> Add All Fields -> New Field:

Name: Summ, Field Type: Calculated, Type: Currency

ЛЕКЦИЯ № 5

1. ФИЛЬТРАЦИЯ И ПОИСК ЗАПИСЕЙ В ТАБЛИЦАХ ДАННЫХ

Фильтрация записей в базе данных позволяет временно делать невидимыми записи, которые не удовлетворяют заданному условию. При включении режима фильтрации база данных будет состоять только из записей, соответствующих заданному условию, это отразится на всех командах поиска, просмотра и перемещения, хотя физически база данных не изменяется на диске.

Для включения режима фильтрации необходимо установить в «True» значение свойства «Filtered» у компонента «Table». Значение «False» свойства «Filtered» выключает режим фильтрации базы данных. Перед включением режима фильтрации необходимо указать условие («фильтр») для отбора записей. Для установки фильтра служит свойства «Filter».

Фильтр может содержать символ «*», означающий любое количество любых символов. Кроме этого в фильтре допус-

каются операторы «AND» и «OR».

Свойство «FilterOptions» служит для установки параметров фильтрации. Значение «*foCaseInsensitive* » служит для выключения (*true=буквы без различия регистра, false=буквы с различием регистра*) различия регистра букв, а значение «*foNoPartialCompare*» служит для включения режима, при котором символ «*» в фильтре является просто символом «*», а не любым количеством любых символов (*false=* знак замены, true=* просто символ*). По умолчанию данные параметры не установлены: регистр букв различается и * – знак подстановки.

Рассмотрим примеры фильтрации данных:

```
// Различение режима букв, * – шаблон
Table1.Open;
Table1.Filtered:=false;
Table1.Filter:='name="Сер*"';
Table1.FilterOptions:=[];
Table1.Filtered:=true;
for i:=1 to Table1.recordcount do begin
 …
end;
```

* * * * * * *

```
// Регистр не различается, * – шаблон
Table1.Filtered:=false;
Table1.Filter:='((name="Сер*") and (Fam="Т_л*")) or (Fam="К*)';
Table1.FilterOptions:=[foCaseInsensitive];
Table1.Filtered:=true;

* * * * * * *

// Регистр различается, * – символ
Table1.Filtered:=false;
Table1.Filter:='corporation="*TSN*"';
Table1.FilterOptions:=[foNoPartialCompare];
Table1.Filtered:=true;
```

2. ФИЛЬТРАЦИЯ ЗАПИСЕЙ ДЛЯ КОМПОНЕНТА «ADOTABLE»

Для включения режима фильтрации необходимо установить в «True» значение свойства «Filtered» у компонента «AdoTable». Значение «False» свойства «Filtered» выключает режим фильтрации базы данных. Перед включением режима фильтрации необходимо указать условие («фильтр») для отбора записей. Для установки фильтра служит свойства «Filter».

Фильтр может содержать символ «%», означающий любое количество любых символов. Кроме этого в фильтре допускаются операторы «AND» и «OR» и оператор «LIKE».

```
AdoTable1.Filtered:=false;
AdoTable1.Filter:='name like "Сер%"';
AdoTable1.Filtered:=true;
for i:=1 to AdoTable1.recordcount do …
 …
```

3. ПОИСК ДАННЫХ В БД

У компонента «Table» имеются специальные команды для поиска нужной записи, удовлетворяющей заданному условию. Наиболее удобные функции поиска – «Locate» и «Lookup». Если поиск для функции «Locate» будет произведен успешно, то найденная запись в БД станет текущей, иначе текущая запись не изменится и ошибка не возникнет. Функция «Lookup» не меняет номера текущей записи при поиске, но если информация не будет найдена, то возникнет ошибка. Функция «Lookup» обычно используется для считывания значения нужного поля после поиска нужной записи, а функция «Locate» для установки новой текущей позиции в БД.

Пример точного поиска (а не фрагмента) с учетом регистра букв:

```
if Table1.Locate( 'famil', 'Petrov', [] ) then s:=Table1 ['name'] else s:='???';
try s:=Table1.Lookup( 'famil', 'Petrov', ‘name’ ); except s:=’???’; end;
```

Пример точного поиска (а не фрагмента) без учета регистра букв:

if Table1.Locate('famil', 'petrov', [loCaseInsensitive]) then s:= Table1 ['name'] else s:='???';

Пример поиска по фрагменту без учета регистра букв:

Table1.Locate('famil', 'P', [loPartialKey, loCaseInsensitive]);

Пример поиска по фрагменту с учетом регистра букв:

if Table1.Locate('Company; Phone', VarArrayOf (['Micro', '408-']), [loPartialKey]) then

s:=Table1['code'] else s:='???';

Пример точного поиска с учетом регистра букв:

try s:=Table1.Lookup ('Company; Phone', VarArrayOf (['Microsoft', '408-431-1000']), 'code');

except s:='???'; end;

ЛЕКЦИЯ № 6

1. РЕЛЯЦИОННЫЙ ДОСТУП К БАЗЕ ДАННЫХ

Компоненты BDE, рассмотренные раннее, позволяют получить доступ к базам данных по технологии, разработанной фирмой Borland под названием Borland Database Engine. Эта технология устарела и поставляется только для совместимости со старыми версиями. Не смотря на это, она хорошо работает со старыми типами баз данных, такими как Paradox и dBase.

На смену BDE технологии пришла DBExpress – это новая технология доступа к данным фирмы Borland. Она отличается большей гибкостью и хорошо подходит для программирования клиент – серверных приложений, использующих базы данных. Компоненты с одноимённой закладки хорошо использовать с базами данных, построенных по серверной технологии, например, Oracle, DB2 или MySQL.

ADO (Active Data Objects) – технология доступа к данным, разработанная корпорацией Microsoft. Очень хорошая

технология, но ее рекомендуется использовать только с базами данных Microsoft, а именно MS Access или MS SQL Server. Её так же можно использовать, если имеется специфичный сервер баз данных, который может работать только через ODBC.

ODBC – это набор драйверов в операционной системе Windows для доступа к базам данных. Через программный интерфейс ODBC приложения могут получать доступ к информации из СУБД, основанных на языке SQL. Для просмотра и конфигурирования ODBC драйверов необходимо вызвать из меню Windows программу администрирования «Источники данных (ODBC)».

Технология ADO использует драйверы Windows ODBC для доступа к БД, поэтому драйвера DBE не требуются. Для работы с базами MS Access лучше всего использовать ODBC-драйвер «Microsoft Jet 4.0 OLE DB Provider.

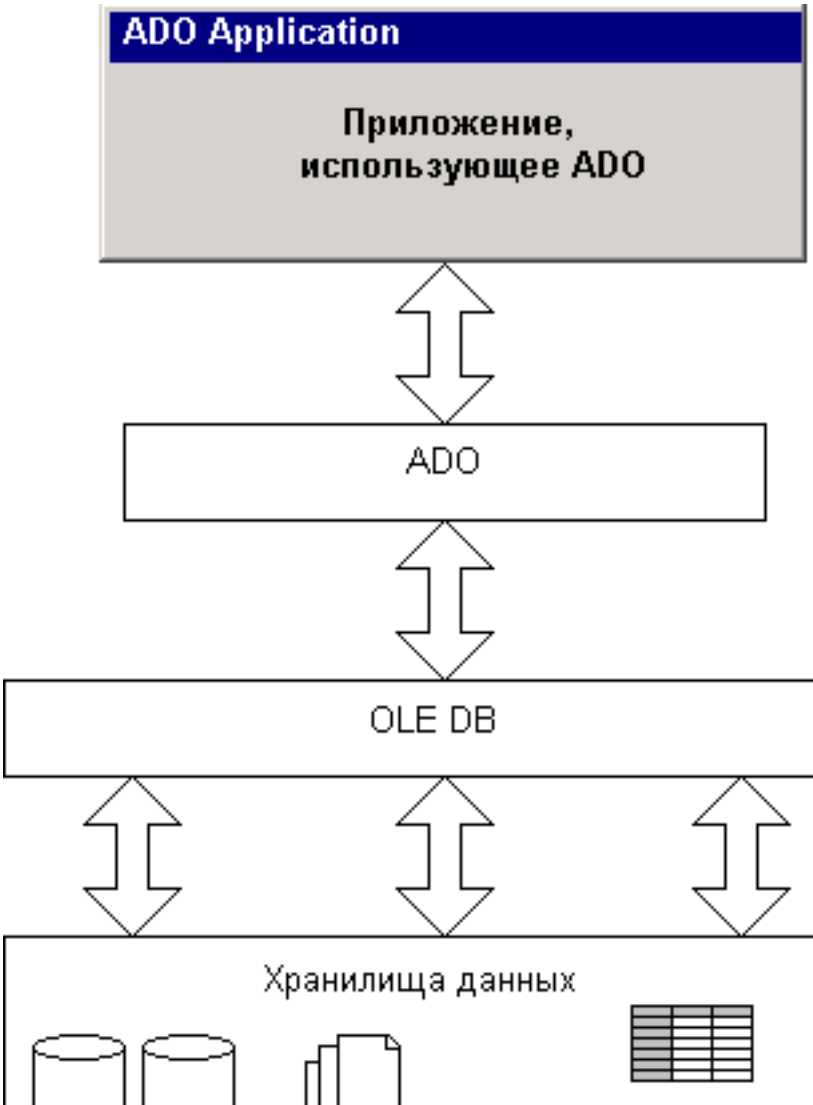

ADO Application
Приложение,
использующее ADO
ADO
OLE DB
Хранилища данных

Рис.1. Схема доступа к данным через ADO

Компонент “Query” является более мощным аналогом компонента «Table» и имеет практически те же свойства, события и методы, что и компонент «Table». Компонент “Query”, в отличие от компоненты «Table», использует для открытия таблицы данных не имя файла и каталога с таблицей, а так называемый SQL-текст с запросом. В тексте SQL-запроса с помощью специальных команд указывается, какой файл нужно открыть, по какому полю его отсортировать, какие поля и записи должны быть видимыми (условия отбора), а какие нет и др.

Компонент “Query” позволяет:

Открывать, просматривать данные и производить поиск в таблицах БД, подобно компоненту «Table»

Создавать виртуальные обобщенные (сцепленные) таблицы из нескольких таблиц данных во время работы программы

Выбирать и находить записи по сложным критериям отбора

Сортировать данные по любым полям без наличия индексных файлов

Группировать данные с нужным условием для подсчета общих сумм, средних значений и др.

Текст SQL-запроса имеет следующий синтаксис:
SELECT [DISTINCT] * или <список_полей>
FROM <список_таблиц>
[WHERE <условие_отбора_записей>]
[ORDER BY <список_полей_для_сортировки>]
[GROUP BY <список_полей_для _ группировки>]
[HAVING <условие_группирования>]
[UNION <вложенный_оператор_SELECT>]

При написании команд в тексте SQL-запроса необходимо следовать той последовательности, которая указаны выше. Регистр символов в тексте запроса не важен. Рассмотрим использование компоненты “Query” и команд SQL-запроса на конкретных примерах.

2. СОЗДАНИЕ ПРОСТЕЙШЕЙ ПРОГРАММЫ ДЛЯ ДОСТУПА К БАЗАМ ДАННЫХ ЧЕРЕЗ ADO

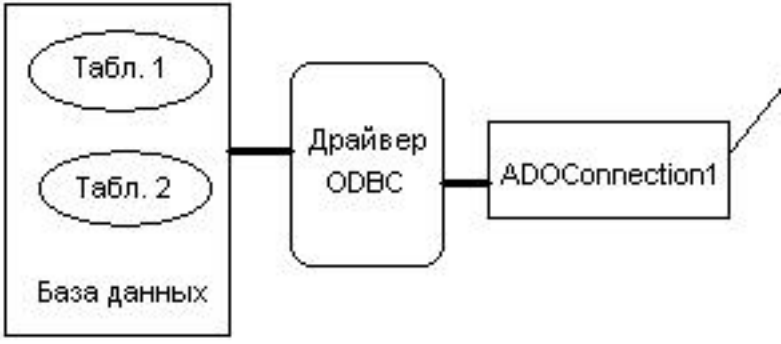


Рис.2. Схема простейшей программы ADO

Простейшая программа состоит из формы, кнопки и следующих компонент:

ADOConnection1 – компонент (из вкладки «ADO») связи с драйвером ODBC и базой данных. Под базой данных понимается настоящая база данных типа MS Access, в которой находится одна или несколько таблиц данных. База данных

MS Access сохраняется на диске в виде одного файла с расширением «mdb»;

ADOTable1 – компонент связи с таблицей данных из базы данных. Данный компонент является аналогом компонента Table из BDE;

DBGrid1 – сетка для отображения и работы с таблицей данных;

DBNavigator1 – навигатор по таблице БД;

DataSource1 – компонент-посредник между компонентами отображения таблицы БД и компонентом ADOTable1.

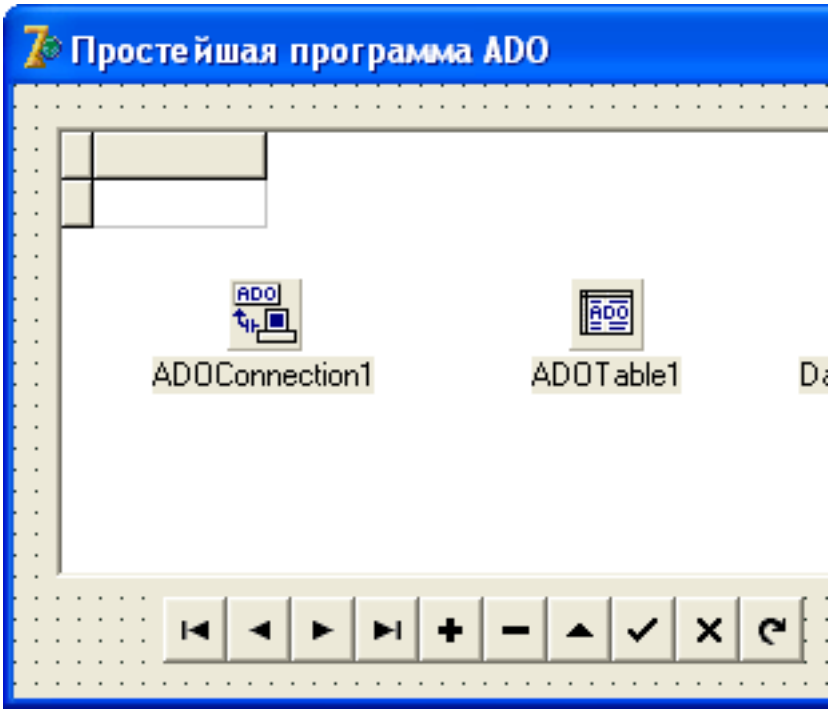


Рис.3. Простейшая программа с ADO-компонентами

```
procedure TForm1.FormCreate(Sender: TObject);
{ Инициализация программы }
begin
   ADOConnection1.ConnectionString:='Data  Source=db/
Database.mdb'; // локальный каталог БД
   ADOConnection1.Provider:='Microsoft.Jet.OLEDB.4.0';//
Имя драйвера доступа к БД
```

```
  ADOConnection1.LoginPrompt:=false; // Отключить запрос имени и пароля доступа к БД
  ADOConnection1.Connected:=true; // Подключаемся к БД

  ADOTable1.TableName:='Справочник'; // Указываем таблицу БД
  ADOTable1.Connection:=ADOConnection1; // Указываем компонент связи с БД

  DataSource1.DataSet:=ADOTable1; // Связываем компоненты DataSource1 и ADOTable1
  DBGrid1.DataSource:=DataSource1; // Связываем компоненты DBGrid1 и DataSource1
  DBNavigator1.DataSource:=DataSource1; // Связываем компоненты DBNavigator1 и DataSource1

  ADOTable1.Open; // Открываем таблицу
 end;

 procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
 { Закрытие программы }
 begin
  ADOConnection1.Connected:=false; // Отключаемся от БД
```

```
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  { Обновление таблицы }
  begin
    ADOTable1.Requery; // Обновляем данные из таблицы
БД
  end;
```

3. ОТКРЫТИЕ ТАБЛИЦЫ ДАННЫХ С ПОМОЩЬЮ КОМПОНЕНТЫ «TQUERY»

Рассмотрим процедуру, открывающую таблицу "c:\db \u2_fakul.dbf". Доступными будут все поля и все записи:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
// Привязываем компонент Datasource1 к компоненту DBGrid1
  DBGrid1.DataSource:=Datasource1;
// Привязываем компонент Query1 к компоненту Datasource1
  DataSource1.DataSet:=Query1;
// Указываем каталог таблиц БД
  DataSource1.DataBaseName:=' c:\student\db';

  with query1 do begin
    Close; // Закрываем (на всякий случай) данные в компоненте Query1
    SQL.Clear; // Очищаем текст SQL-запроса
// Занесем в компонент query1 текст SQL-запроса
    SQL.Add('select *'); // даем команду выбрать все поля
```

```
    SQL.Add('from "u2_fakul.db"'); // указываем имя таблицы БД

    Open; // Открываем таблицу данных "c:\db\u2_fakul.db"
   end;
 end;
```

Рассмотрим фрагмент процедуры, открывающий таблицу "c:\db\u2_fakul.dbf", причем записи будут доступные все, а поля только с именами «name» и «kod_fakul». Все записи отсортируются по полю «name» в возрастающем порядке.

```
 with query1 do begin
   Close; SQL.Clear;
   SQL.Add('select name, kod_fakul');
   SQL.Add('from "c:\db\u2_fakul.dbf"');
   SQL.Add('order by name');
   Open;
  end;
```

Рассмотрим фрагмент процедуры, открывающий таблицу "c:\db\u2_fakul.dbf", причем записи будут доступные все, а поля только с именами «name» и «kod_fakul». Все записи отсортируются по полю «kod_fakul» в возрастающем порядке и по полю «name» в убывающем порядке. В тексте SQL-запроса используются две строковые переменные s1 и s2.

```
 var s1,s2: string;
```

```
begin
with query1 do begin
   s1:='name, kod_fakul'; s2:='c:\db\u2_fakul.dbf';
   Close; SQL.Clear;
   SQL.Add('select '+s1); SQL.Add('from "'+s2+'"');
   SQL.Add('order by kod_fakul, name DESC');
   Open;
  end;
end;
```

После открытия данных командой «Open» текст SQL-запроса не имеет значения, и его изменение не изменит отображаемых компонентом данных. Для активизации нового текста SQL-запроса необходимо закрыть данные компонента “Query” командой «Close», после чего задать новый текст запроса и открыть данные командой «Open».

Программирование с использованием SQL-запросов является наиболее передовой технологией работы с базами данных. Данная технология называется реляционным способом доступа к базе данных. Непосредственный доступ к базе данных через компонент «Table» называется навигационным способом. Навигационный способ является устаревшим, т.к. он рассчитан на использование локальных базам данных без сетевой поддержки.

При работе с распределенными в сети (коллективными)

базами данных используется только реляционный способ, т.к. только он позволяет предотвращать сетевые коллизии при одновременном доступе нескольких пользователей к базе данных, и существенно разгружает сетевой трафик. Поэтому рекомендуется использовать компонент “Query” вместо компоненты «Table», хотя нужно отметить, что навигационный способ обычно работает быстрей, чем реляционный.

Использование реляционного способа доступа к БД через компоненту “Query” позволяет не только легко производить сложнейшую обработку нескольких взаимосвязанных таблиц данных, но и производить модификацию данных (вставлять новые записи, производить групповое удаление данных, делать каскадную замену информации и др.).

Пример отбора записей по значениям символьного поля:

```
SELECT Name FROM Pers WHERE Post='Менеджер'
  // А лучше так:
      SELECT Name FROM Pers WHERE
UPPER(TRIM(Post))='МЕНЕДЖЕР'
```

Использование псевдонимов:

```
SELECT P.Name FROM “People.db” As P
```

Пример сцепления двух таблиц:

```
SELECT p.fio,p.tel
FROM “tel.db” as t, “people.db” as p
```

```
WHERE t.code_p =p.code_p;
```

ЛЕКЦИЯ № 7

1. ОСНОВНЫЕ СОБЫТИЯ КОМПОНЕНТ ДОСТУПА К ТАБЛИЦАМ ДАННЫХ

Компонент “Table” имеет множество событий, которые возникают при любой работе с базой данных, будь то открытие БД, закрытие, переход на другую запись, удаление записи и т.д. Данные события могут возникнуть в одинаковой степени при работе с любым визуальным компонентом, связанным с базой данных. Основные события компонента следующие:

Имя события	Событие возникает
AfterOpen / BeforeOpen	После / перед открытием базы данных (например, для метода Open)
AfterClose / BeforeClose	После / перед закрытием базы данных (например, для метода Close)

AfterScroll / BeforeScroll	После / перед переходом на другую запись в БД (например, для метода Next)
AfterEdit / BeforeEdit	После / перед началом редактирования текущей записи (например, для метода Edit)
OnEditError	В случаи возникновения ошибки при попытке начать редактирование текущей записи (например, для метода Edit)
AfterInsert / BeforeInsert / OnNewRecord	После / перед / в процессе вставки или добавления в БД новой записи (например, для метода Insert)
AfterPost / BeforePost	После / перед сохранением сделанных изменений в текущей записи (например, для метода Post)
OnPostError	В случаи возникновения ошибки при сохранении сделанных изменений в текущей записи (например, для

	метода Post)
AfterDelete / BeforeDelete	После / перед удалением текущей записи (например, для метода Delete)
OnDeleteError	В случаи возникновения ошибки при удалении текущей записи (например, для метода Delete)
AfterCancel / BeforeCancel	После / перед отменой сделанных изменений в текущей записи (например, для метода Cancel)

Контроль ввода данных перед сохранением записи:

```
procedure Table_FakulBeforePost(DataSet: TDataSet);
var s: string;
begin
  try s:=table1['name']; except s:=''; end;
  if s='' then begin
    MessageDlg('Укажите имя!', mtInformation, [mbOK],0);
abort;
    end;
end;
```

ЛЕКЦИЯ № 8

1. ПРИМЕРЫ МЕТОДОВ РАБОТЫ С ДАННЫМИ БД

1. Использование закладок:

```
procedure TForm1.Button1Click(Sender: TObject);
var SaveP: TBookmark;
begin
try
  SaveP := Table1.GetBookmark;
……….
  Table1.GotoBookmark(SaveP);
  Table1.FreeBookmark(SaveP);
except end;
end;
```

2. Показ количества записей в БД:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  try
          StaticText1.caption:='Всего доступно записей:
```

```
'+inttostr(Table1.recordcount);
    except; end;
    Application.ProcessMessages;
  end;
```

3. Изменение значений поля через SQL-запрос:

```
  UPDATE Pers.db
     SET Salary = Salary + 50
      WHERE Salary < 500 // Если сотрудник имеет оклад менее 500 руб., то его оклад увелич. на 50 руб.
  ……
  UPDATE Store.db
        SET S_Price = S_Price * 1.1 // Цена всех товаров увеличивается на 10 %
```

4. Добавление записи через SQL-запрос:

```
  INSERT INTO Store.db
      (Name, Price) VALUES (“Морковь”, 4.7)
```

5. Добавление нескольких записей через сложный SQL-запрос:

```
  INSERT INTO CardsArchives (Code, Move, Date)
      SELECT C_Code, C_Move, C_Date
        FROM Cards WHERE C_Date BETWEEN 1.1.98 AND 31.12.98
```

6. Удаление записей через SQL-запрос:

```
DELETE FROM Store.db
    WHERE S_Quantify = 0
```

7. Использование параметров в SQL-запросах (свойство «Params» типа «TParams»):

```
Query1.ParamByName('pCena').AsFloat:= StrToFloat(Edit1.Text);
```

или

```
Query1.Params[0].AsFloat:= StrToFloat(Edit1.Text);
```

Пример с параметром:

```
SELECT Name, Post, Cena
    FROM Pers.db
    WHERE Cena >= :pCena
```

2. ИСПОЛЬЗОВАНИЕ SQL-ФУНКЦИЙ ДЛЯ РАБОТЫ С БД

1. Сроковые функции SQL:
UPPER(Str)
LOWER(Str)
TRIM(Str)
SUBSTRING(Str FROM n1 TO n2)
CAST(<Expression> AS <Type>) – приведение выражения Expression к типу Type
|| – конкатенация строк

Например:

```
SQL.Add('Select cast(ff.nomer as numeric) as nomer, kk.famil,');
SQL.Add('cast((SUBSTRING(kk.ima FROM 1 FOR 1)||');
SQL.Add('SUBSTRING(kk.otch FROM 1 FOR 1)) as character(2)) as io');
```

2. Создание таблицы:

```
CREATE TABLE NewTable.db
  (Number INTEGER,
   Name CHAR(20),
   Birthday DATE);
```

3. Удаление таблицы БД:
DROP TABLE NewTable.db

4. Изменение структуры таблицы:
ALTER TABLE Pers.db
ADD Section SMALLINT, //добавление поля
ADD Note CHAR(30),
DROP Post // удаление поля

5. Создание и удаление индекса:
CREATE INDEX indName ON Personnel.db (Name)

6. Удаление индекса:
DROP INDEX “personnel.db”.indName

7. Задание вычисляемого поля:
SELECT “– ” || Name, Salary, Salary*1.1
FROM Personnel;

Выводятся старые значения окладов сотрудников и новые, увеличенные на 10%. К каждой фамилии с помощью операции конкатенации добавляется символ “-”.

8. Отбор записей с уникальными значениями поля:
SELECT DISTINCT Post FROM Personnel

9. Проверка частичного совпадения значения поля:

```
SELECT Name
    FROM Perssonel
    WHERE Name LIKE “Ав”
```

10. Проверка частичного совпадения с помощью шаблона:

```
SELECT Name
    FROM Goods
    WHERE Name LIKE “%” || “п_р” || “%”
```

% – замещение любого кол-ва символов, в том числе и нулевого

_ – замещение одного символа

11. Проверка нулевых значений поля

```
SELECT *
    FROM Store
    WHERE S_Price IS NULL
```

12. Проверка вхождения записи в список:

```
SELECT Name, Salary
    FROM Perssonel
    WHERE Post IN (“Менеджер”,”Ст. менеджер”)
```

13. Проверка вхождения записей в диапазон:

```
SELECT *
    FROM Cards
    WHERE C_Date BETWEEN “13.4.99” AND “15.4.99”
```

14. Группирование записей. При группировании записей автоматически исключается повтор значений группируемых полей:

```
SELECT C_Date, COUNT (C_Date)
    FROM Cards
    GROUP BY C_Date
    HAVING COUNT(C_Date)>50
```

Выводятся данные для тех периодов времени, когда движение товара было интенсивным, т.е. общее число записей в таблицу превышало 50. В группировке могут использоваться следующие функции:

AVG() – среднее значение
MAX()
MIN()
SUM()
COUNT() – кол-во значений
COUNT(*) кол-во ненулевых значений

15. Сортировка по двум полям:

```
SELECT Name, Post, Salary
    FROM Pers.db
    ORDER BY Post, Salary DESC
```

16. При внешнем соединении таблиц можно указать, ка-

кая из таблиц будет главной, а какая – подчиненной. В этом случае формат операнда FROM имеет вид:

FROM <Таб-ца1> [<Вид соединения>] JOIN <Таб-ца2> ON <Условие отбора>

Вид соединения, какая из 2 таблиц будет главной:

LEFT – слева

RIGHT – справа

{ Left join – таблицы объединяются, левая таблица – основная }

{ Пример ниже выводит номера людей и тех, у кого нет телефона }

```
SELECT distinct p.Fam, p.Name, pNumber
FROM “c:\mydb\people.dbf” as p LEFT JOIN “c:\mydb\tel.dbf” as t
ON (p.ID_People=t.ID_People)
```

{ Right join – таблицы объединяются, правая таблица – основная }

{ Если в предыдущем примере заменить LEFT JOIN на RIGHT JOIN, то программа выведет все номера телефонов из базы «Tel», и соответствующую телефону фамилию, если она имеется }

{ Full join – таблицы объединяются с дополнениями по

каждой таблице }

{ Если в предыдущем примере заменить LEFT JOIN на FULL JOIN, то программа выведет все номера телефонов и все фамилии, причем какая-либо фамилия может быть без телефона, и какой-нибудь телефон – без фамилии }

ЛЕКЦИЯ № 9

1. МЕХАНИЗМ ТРАНЗАКЦИЙ ПРИ ОБРАБОТКЕ БАЗ ДАННЫХ

1. Транзакция – это действия на таблицами баз данных, которые должны быть выполнены полностью, от начала до конца. Если окажется, что действия над таблицами БД в транзакции не могут быть полностью и нормально выполнены, то необходимо их (проделанные действия) полностью отменить и вернуть таблицы данных в исходное состояние (до начала действий транзакции).

2. В Delphi работа с транзакциями обычно организуется через компонент типа «TDatabase». Для транзакции существует команда начала транзакции «StartTransaction», подтверждения транзакции «Commit» и отмены (отката) транзакции «Rollback» (в случае ошибки транзакции).

3. Алгоритм работы с транзакциями для таблиц типа Paradox:

Закрываем все таблицы из каталога транзакций, иначе старт транзакции будет невозможен;

Настраиваем режим транзакции на таблицы типа Paradox;

Указываем каталог транзакции компоненту DataBase;

Открываем нужные таблицы для работы;

Стартуем транзакцию;

Делаем нужную работу с БД (например, каскадное удаление);

Подтверждение транзакции;

Отмена транзакции при проблеме (при каком-либо сбое в пунктах 6 или 7);

Обновляем таблицы данных, задействованных в транзакции.

4. Рассмотрим пример работы с транзакциями:

```
procedure TForm1.Button1Click(Sender: TObject);
var i: word;
begin
  try
    // Закрываем все таблицы из каталога транзакций
    // иначе старт транзакции будет невозможен
    try Table1.Close; Table2.Close; except end;

     Database1.TransIsolation:=tiDirtyRead; // Настраиваем режим транзакции на таблицы типа Paradox
    Database1.DataBaseName:='C:\Student\Db'; // Указываем каталог транзакции
     Table1.Open; Table2.Open; // Открываем нужные таблицы для работы
     Database1.StartTransaction; // Стартуем транзакцию
```

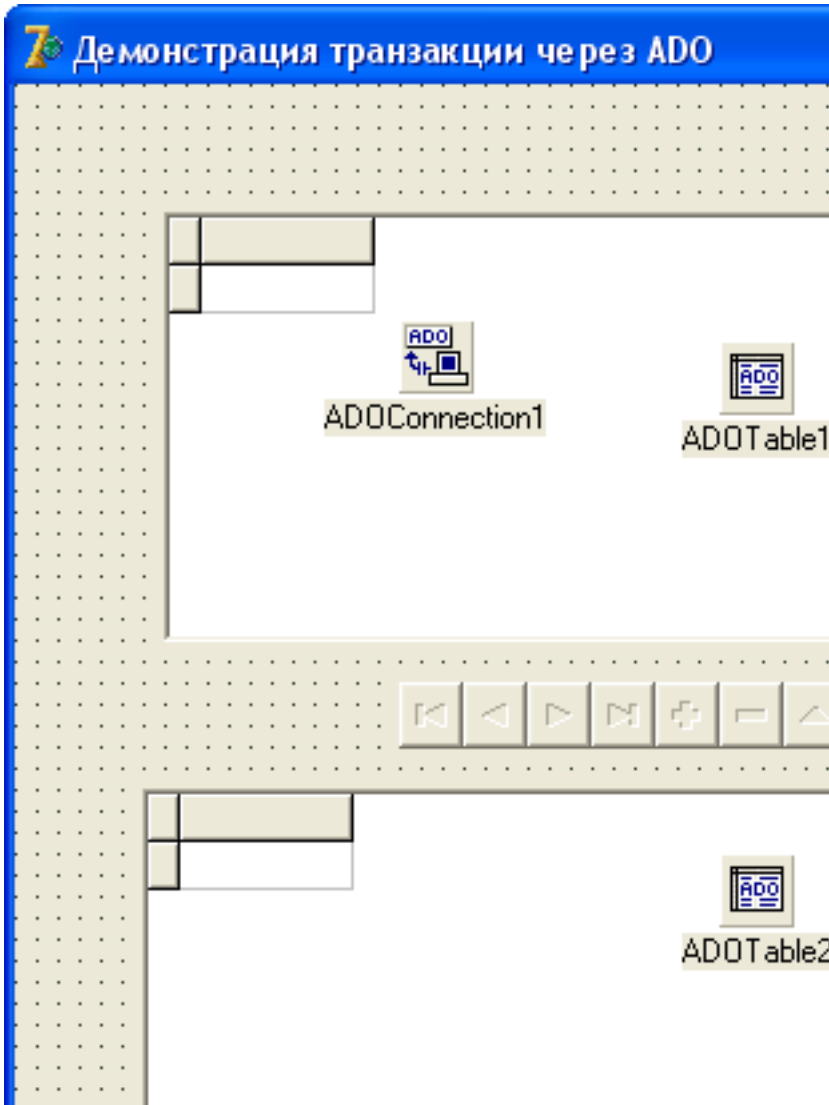
```
    // Делаем работу с БД –
                Table1.Delete; Table1.Delete; Table2.Delete;
Table2.Delete; showmessage('!!!');
          abort; // Имитируем проблему
    // Конец работы с БД –

     Database1.Commit; // Подтверждение транзакции
   except
    Database1.Rollback; // Отмена транзакции при проблеме
   end;

    // Обновляем таблицы данных
  // Вместо «Table1.Refresh; Table2.Refresh;» лучше так:
        for i:=0 to Database1.DataSetCount-1 do
Database1.DataSets[i].Refresh;
  end;
```

2. ТРАНЗАКЦИИ В ТЕХНОЛОГИИ ADO

```
procedure TForm1.Button1Click(Sender: TObject);
var i: integer;
begin
  ADOConnection1.BeginTrans; // Начало транзакции
  try
    ADOTable1.Delete;
    ADOTable2.Delete; ADOTable2.Delete;
      showmessage('А сейчас будет имитация сбоя и откат транзакции!');
    abort; // Генерация сбоя
    ADOConnection1.CommitTrans; // Подтверждение транзакции
    except
    ADOConnection1.RollbackTrans; // Откат транзакции
  end;
// ADOTable1.Requery; ADOTable2.Requery; // Обновление данных из таблицы
  // Обновление данных из всех таблиц БД (так лучше)
   for i:=0 to ADOConnection1.DataSetCount-1 do
     ADOConnection1.DataSets[i].Requery;
end;
```


Демонстрация транзакции через ADO
ADOConnection1
ADOTable1
ADOTable2

3. КАСКАДНОЕ УДАЛЕНИЕ ДАННЫХ С БД

Каскадное удаление данных – это удаление связанных между собой данных с разных таблиц одновременно.

```
procedure TForm1.Button6Click(Sender: TObject);
var k: integer;
begin
  ADOTable1.Filtered := False;
  k:=ADOTable2['Kod_firmi'];
  ADOTable2.Delete;
  ADOTable1.Filter := 'Kod_postavschika = '+IntToStr(k);
  ADOTable1.Filtered := True;
  ADOTable1.First;
  while not ADOTable1.Eof do ADOTable1.Delete;
  ADOTable1.Filtered := False;
                DBLookupComboBox3.KeyValue :=
ADOTable2['Nazv_firmi'];
end;

procedure TForm1.ADOTable2AfterScroll(DataSet:
TDataSet);
begin
DBLookupComboBox3.KeyValue :=
```

```
ADOTable2['Nazv_firmi'];
  end;

  ********

  procedure TForm1.Button7Click(Sender: TObject);
  begin
  if Edit5.Text = '' then
    ShowMessage('Сначала нужно ввести телефон директо-
ра.')
  else
    begin
      ADOConnection1.BeginTrans;
      try
        ADOQuery4.Close;
        ADOQuery4.SQL.Clear;
        ADOQuery4.SQL.Add('delete');
        ADOQuery4.SQL.Add('from Diski_CD');
            ADOQuery4.SQL.Add('where Kod_postavschika in
(select Kod_firmi from Firmi_postavschiki where Tel_dir like
'''+Edit5.Text+'%'')');
        ADOQuery4.ExecSQL;

        ADOQuery4.Close;
        ADOQuery4.SQL.Clear;
        ADOQuery4.SQL.Add('delete');
```

```
        ADOQuery4.SQL.Add('from Firmi_postavschiki');
        ADOQuery4.SQL.Add('where Tel_dir like '"+Edit5.Text
+'%'");
        ADOQuery4.ExecSQL;

        ADOConnection1.CommitTrans;
      except
        ADOConnection1.RollbackTrans;
       ShowMessage('Произошла ошибка удаления, удаление
записей отменено.');
      end;

  *******

  procedure TForm1.Button1Click(Sender: TObject);
  var i:integer;
  begin

  ADOConnection1.BeginTrans;
  try
  ADOQuery4.Close;
  ADOQuery4.SQL.Clear;
  ADOQuery4.SQL.Add('Delete ');
  ADOQuery4.SQL.Add('From disc_1');
  ADOQuery4.SQL.Add('where cod_firm in (select cod_firm
from firm_2');
```

```
ADOQuery4.SQL.Add('Where firm_2.cod_city in (Select cod_city_proiz from city_3');
ADOQuery4.SQL.Add('Where (name_city like "%-%")and not(name_city like "%,%")))');
ADOQuery4.ExecSQL;

ADOQuery4.SQL.Clear;
ADOQuery4.SQL.Add('Delete');
ADOQuery4.SQL.Add('From firm_2');
ADOQuery4.SQL.Add('where firm_2.cod_city in (select cod_city_proiz from city_3');
ADOQuery4.SQL.Add('Where (name_city like "%-%")and not(name_city like "%,%"))');
ADOQuery4.ExecSQL;

try
for i:=0 to ADOConnection1.DataSetCount do
ADOConnection1.DataSets[i].Requery;
except

end;

ADOConnection1.CommitTrans;
except
ADOConnection1.RollbackTrans;
ShowMessage('Произошла ошибка удаления, удаление за-
```

```
писей отменено.');
    end;

    end;
```

ЛЕКЦИЯ № 10

1. СОЗДАНИЕ И РАБОТА С ОТЧЕТАМИ ДАННЫХ БД

1.1. Создание простейших отчетов бд через HTML

```
Uses …, shellapi;

procedure TForm1.Button8Click(Sender: TObject);
var tst: TStringList;
begin
tst:=TStringList.Create;
tst.Add('<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.0 Transitional//EN" >');
tst.Add('<HTML>');
tst.Add('<HEAD>');
tst.Add('<title>Пробная страничка</title>');
tst.Add('</HEAD>');
tst.Add('<BODY>');
tst.Add(DataSetTableProducer1.Content);
tst.Add('</BODY></HTML>');
tst.SaveToFile('htmldoc.html');
```

```
tst.Free;
ShellExecute(handle, 'open', 'htmldoc.html', nil, nil,
SW_SHOWNORMAL);
end;
```

1.2. Создание простейших отчетов БД с помощью QReport

Для рассмотрения вопроса создания простейшего отчета БД необходимо сделать две формы. На первой форме необходимо:

нанести компонет Table1 и Button1;

настроить Table1 на конкретную таблицу данных и открыть ее;

сдетать следующую процедуру обработчик для кнопки:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  form2.QuickRep1.Preview;
end;
```

На второй форме необходимо:

подключить в разделе «Uses» модуль «unit1»;

нанести компонет «QuickRep1» из вкладки «QReport» палитры компонент. Настроить его свойства;

нанести на компонет «QuickRep1» **шесть** компонент типа

QRBand;

настроить компоненты QRBand на нужные модификации;

нанести в нужные места отчета компоненты QRLabel, QRDBText, QRSysData и настроить их свойства;

QuickRep

Базовый компонент для создания отчетов. Пустая форма отчета.

Свойство	Назначение свойства
DataSet	Привязка к таблице БД, для которой делается отчет. Это свойство должно быть корректно задано для нормальной работы отчета

QRBand

Из этих компонентов-контейнеров стоится скелет отчета. Отчет обычно состоит как минимум из шести таких компонентов-контейнеров. Компонент имеет множество модификаций, от которых зависит назначение компонента

Свойство BandType	Назначение компонента
rbTitle	верхняя шапка всего отчета, выдается вверху первой страницы

rbSummary	нижняя шапка всего отчета, выдается внизу последней страницы
rbPageHeader	верхняя шапка для каждой страницы (включая первую)
rbPageFooter	нижняя шапка для каждой страницы (включая последнюю)
rbColumnHeader	верхняя шапка для области вывода данных, выдается на каждой странице перед данными
rbDetail	контейнер для компонент вывода данных из БД, область данных
rbGroupFooter	контейнер для компонент вывода данных БД для группировки

QRLabel

Отображает текстовые данные в указанном месте отчета. Эквивалент обычного «Label»

Свойство	Назначение свойства
Caption	комментарий (слева) к отоб-

	ражаемым данным

QRDBText

Отображает данное из поля БД в указанном месте отчета. Эквивалент обычного «DBEdit». Компонент наносится обычно на контейнер «QRBand->rbDetail »

Свойство	Назначение свойства
Aligment	taCenter, taLeftJustify, taRightJustify
DataSet	указание источника данных (ссылка на Table или Query). Данное значение обязательно должно быть точно такое же, как и свойство «DataSet» у компонента QuickRep
DataField	имя поля таблицы БД

QRSysData

Компонент для отображения системных данных (текущего времени, даты, номера страницы и др.)

Свойство Data	Назначение компонента
qrsDateTime	системная дата и время
qrsPageNumber	номер печатаемой страницы
qrsDetailNo	порядковый номер записи

	БД из области данных (по всему отчету)
	в событии «OnPrint» можно написать: **Value:=inttostr(Form1.Table**
qrsDetailCount	количество отображаемых в отчете записей БД (по всему отчету)
rbGroupFooter	нижний колонтитул группы. Контейнер для отображения значений суммы и др. по группировке данных

Свойство	Назначение свойства
Text	комментарий (слева) к отображаемым данным

QRShape

Компонент для рисования простейшей фигуры

Свойство Shape	Назначение компонента
qrsHorLine	горизонтальная линия
qrsVertLine	вертикальная линия
qrsRectangle	Четырехугольник

QRExpr

Компонент отображения вычисляемого значения. Очень мощный компонент. Обычно располагается на контейнерах QRGroup, QRBand (вида rbGroupFooter) и др.

Свойство	Назначение свойства
Expression	выражение отображения, например:
	sum(query1.zarpl)
	query1.fam +' ('+query1.name+')'
	IF(query1sc<>0,100*((sum(q query1.so)),0)
	upper(query1.famil+' '+COPY(query1.ima, 1, 1)+'. '+COPY(query1.otch, 1, 1)+'.')
Mask	маска отображения значения, например '### ### ### ### ### ### ##0.00' или '**##**0.0%'
Master	Ссылка на компонент

	QuickRep, например «QuickRep1»

QRGroup

Компонент контейнер для группировки данных БД по полю.

Свойство	Назначение свойства
Expression	Имя поля группировки, например «Table1.group». Выбранная таблица группировки должна быть открыта с индексом по такому же полю
Footerband	Ссылка на компонент «QRBand-типа rbGroupFooter»