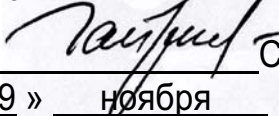


УТВЕРЖДАЮ

Декан АВТФ


С.А. Гайворонский
« 19 » ноября 2008 г.

Е.А. Мирошниченко, Н.А. Шестаков

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

ЦИКЛ ЛАБОРАТОРНЫХ РАБОТ

Часть I

Методические указания к циклу лабораторных работ
по дисциплине «Проектирование информационных систем»
для магистрантов, обучающихся по магистерской программе
«Компьютерный анализ и интерпретация данных»
направления 230100 «Информатика и вычислительная техника»

Издательство
Томского политехнического университета
2008

УДК 621.391.001(076.5)

ББК 32.973.202я73

М64

Мирошниченко Е.А.

М64

Проектирование информационных систем. Цикл лабораторных работ. Часть I: методические указания к циклу лабораторных работ по дисциплине «Проектирование информационных систем» для магистрантов, обучающихся по магистерской программе «Компьютерный анализ и интерпретация данных» направления 230100 «Информатика и вычислительная техника» / Е.А. Мирошниченко, Н.А. Шестаков. – Томск: Изд-во Томского политехнического университета, 2008. – 33 с.


ISBN 5-98298-352-7

УДК 621.391.001(076.5)

ББК 32.973.202я73

Методические указания рассмотрены и рекомендованы
к изданию методическим семинаром кафедры
вычислительной техники АВТФ
10 сентября 2008 г.

Зав. кафедрой ВТ
доктор технических наук,
профессор

 Н.Г. Марков

Председатель учебно-методической
комиссии

 В.И. Рейзлин

Рецензент

Кандидат технических наук,
доцент кафедры программирования ФПМК ТГУ

С.А. Останин

ISBN 5-98298-352-7

© Мирошниченко Е.А., Шестаков Н.А., 2008

© Томский политехнический университет, 2008

© Оформление. Издательство Томского
политехнического университета, 2008

ВВЕДЕНИЕ

Предлагаемое методическое пособие предназначено для магистрантов направления «Информатика и вычислительная техника» по магистерской программе «Компьютерный анализ и интерпретация данных» факультета автоматизации и вычислительной техники. Пособие используется при выполнении лабораторных работ и индивидуальных заданий по дисциплине «Проектирование информационных систем».

Данное пособие не вполне обычно в том смысле, что оно не нацелено на описание конкретных лабораторных работ и способов их выполнения. Дело в том, что излагаемый в пособии материал используется студентами при выполнении *практически всех работ и заданий учебной дисциплины*.

Методическое пособие полностью посвящено современным средствам и технологиям, на которых строится практическая часть дисциплины, а именно средствам разработки приложений баз данных Borland/CodeGear Delphi 2007. Получаемые студентами навыки работы составят тот практически ценный багаж, с которым выпускники придут к современному работодателю.

Необходимость в данном пособии продиктована тем, что самостоятельное освоение студентами этих продуктов даже на минимально приемлемом уровне требует значительного времени и дорогостоящей литературы. Методическое пособие предлагает минимальный набор сведений и указаний, который позволяет студенту достаточно быстро начать работу, приступить к выполнению лабораторных работ и индивидуальных заданий по учебной дисциплине.

1. АРХИТЕКТУРА СИСТЕМЫ БАЗ ДАННЫХ

В настоящее время термины «информационная система» и «система баз данных» фактически слились по смыслу и области применения. Поэтому на практике они могут использоваться как взаимозаменяемые, хотя первый, вообще говоря, шире второго.

Самое простое (и даже «грубое») определение термина «система баз данных» дал авторитетный специалист в области баз данных К. Дж. Дейт: *система баз данных – это компьютерная система хранения записей.*

Одно из наиболее простых и удачных определений термина «информационная система» таково: *информационная система – это система, которая доставляет нужную информацию нужному адресату в требуемом виде.*

Система баз данных состоит из:

- аппаратной части: компьютеры, средства хранения, средства связи и т. д.
- программной части: клиентские приложения, системы управления базами данных (СУБД) и т. д.

Клиентские приложения в составе системы баз данных часто называют просто *приложения баз данных.*

На этапе конструирования системы баз данных необходимо:

- выбрать СУБД
- выполнить проектирование базы данных
- создать базу данных в соответствии с проектом
- выполнить проектирование необходимых клиентских приложений и реализовать их.

Данное пособие помогает начинающему приступить к реализации клиентских приложений в среде *CodeGear Delphi 2007*. Впрочем, тот же материал является актуальным и для предыдущих версий *Borland/CodeGear Delphi*.

Традиционно под архитектурой системы баз данных понимают состав и организацию элементов системы, где элементы рассматривают на уровне программных компонентов.

Главные компоненты системы баз данных таковы:

- база данных (одна или несколько);
- СУБД;
- приложение баз данных (одно или несколько).

По степени распределённости архитектуры систем баз данных делят на:

- настольные (локальные);
- распределённые.

В настольной (*desktop*) системе все компоненты находятся и функционируют на одном компьютере. В распределённой системе различные компоненты находятся и функционируют, в общем случае, на разных компьютерах, через некоторую среду передачи данных.

Распределённые системы делят на два вида архитектур:

- файл-серверные;
- клиент-серверные.

В системах с архитектурой файл-сервер база данных находится на некотором файловом сервере, а на рабочих станциях находятся клиентские приложения и экземпляры СУБД. Каждая СУБД управляет одной и той же БД по сети.

В системах с архитектурой клиент-сервер существует сервер, на котором находится БД и СУБД, а на рабочих станциях находятся клиентские приложения. Клиентские приложения взаимодействуют с единой СУБД по сети, а СУБД управляет БД непосредственно, в монопольном режиме.

Клиент-серверные системы делят на два вида архитектур:

- двухзвенные;
- многозвенные.

В двухзвенной системе существует всего два типа «звеньев»: сервер БД (с СУБД и БД) и рабочая станция (с клиентскими приложениями). Клиентские приложения взаимодействуют непосредственно с СУБД.

В многозвенной системе существует три типа «звеньев»: сервер БД (с СУБД и БД), обычная рабочая станция (с клиентскими приложениями) и промежуточный сервер приложений (*application server*). Клиентские приложения не взаимодействуют с СУБД непосредственно. Вместо этого они взаимодействуют с серверами приложений. Сервера приложений, в свою очередь, взаимодействуют непосредственно с СУБД или с другими серверами приложений.

В данном пособии рассматриваются системы наиболее широко распространённой архитектуры, а именно *двухзвенные клиент-серверные системы*.

2. АРХИТЕКТУРА КОМПОНЕНТОВ БАЗ ДАННЫХ VCL

Для создания приложений баз данных в среде Delphi надо понять общие принципы использования компонентов VCL, предназначенных для этих целей.

Компоненты VCL для создания приложений БД делятся на два вида: *невизуальные* и *визуальные*.

Невизуальные компоненты обеспечивают соединение с СУБД, выполнение запросов, хранение в памяти всех необходимых данных и выполнение необходимых операций. Как следует из их названия, это компоненты не занимаются отображением информации и не видны пользователю. Они доступны только программисту.

Визуальные компоненты – это средства пользовательского интерфейса, с помощью которых пользователь просматривает данные, непосредственно редактирует их и отдаёт необходимые команды. Сами по себе визуальные компоненты с СУБД не взаимодействуют, они взаимодействуют с невидимыми компонентами.

Таким образом, «цепочка» от пользователя к БД выглядит следующим образом: пользователь ↔ визуальные компоненты ↔ невидимые компоненты ↔ СУБД ↔ БД.

Компании Microsoft и Borland приложили немало усилий для создания универсальных технологий доступа к БД из клиентских приложений. Проблема состоит в том, что существует множество различных СУБД с одной стороны и множество различных средств разработки с другой стороны.

За последние десятилетия были предложены несколько технологий и эволюция продолжается. Дадим поверхностный обзор этих технологий, поскольку даже устаревшие средства ещё используются.

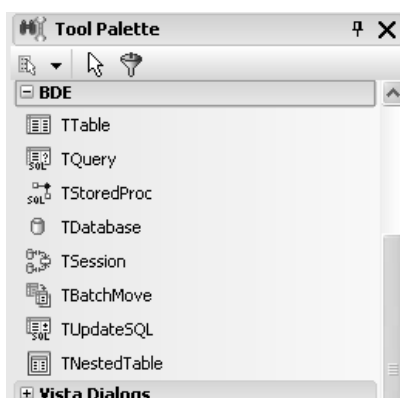


Рис. 1. Невизуальные компоненты для работы с BDE

Компания Borland предложила технологию BDE (Borland Database Engine), которая предназначалась, главным образом, для средств разра-

ботки от Borland. В настоящее время компания объявила BDE устаревшей технологией, которая не развивается, но поддерживается. Тем не менее, принципы, которые компания Borland опробовала на BDE, продолжают составлять основу внутренней архитектуры компонентов VCL. Невизуальные компоненты для работы через BDE находятся на вкладке BDE палитры инструментов (Tool Palette), как показано на рис. 1.

ODBC (Open Database Connectivity) – это первая технология доступа к данным компании Microsoft. ODBC входит в любую редакцию ОС Microsoft Windows. Однако (как и BDE) технология ODBC официально объявлена компанией Microsoft устаревшей технологией, которая не развивается, но поддерживается.

OLE DB (Object Linking and Embedding for Databases) – технология Microsoft, основанная на OLE/COM, которая пришла на смену ODBC. OLE DB позволяет работать не только с СУБД, но и с другими источниками данных: файлами, электронными таблицами, электронной почтой.

dbExpress – новая технология от Borland/CodeGear, особенностью которой является упрощённая, но очень быстрая работа с источниками данных. Невизуальные компоненты для работы через dbExpress находятся на вкладке dbExpress палитры инструментов (рис. 2).

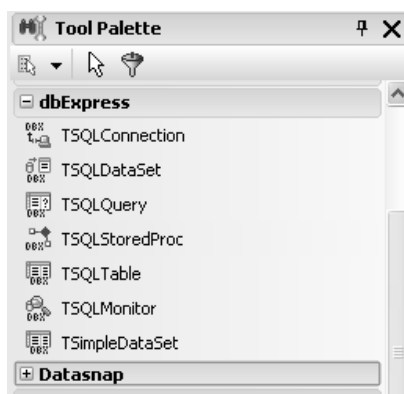


Рис. 2. Невизуальные компоненты для работы с dbExpress

ADO (ActiveX Data Objects) обеспечивает более дружелюбную оболочку для базовой технологии OLE DB. OLE DB находится на более низком уровне программирования, а ADO предоставляет объектно-ориентированный интерфейс более высокого уровня. Невизуальные компоненты для работы через ADO находятся на вкладке dbGo палитры инструментов¹ (рис. 3). Именно ADO долгое время является основной технологией доступа, которую используют разработчики под Windows. Поэтому в данном пособии внимание будет уделено именно ADO.

¹ В прежних версиях Delphi эта вкладка называлась ADO.

Общим термином MDAC (Microsoft Data Access Components) обозначается пакет компонентов доступа к данным Microsoft, который распространяется вместе с Microsoft Windows. MDAC включает в себя ODBC, OLE DB, ADO и RDS (Remote Data Services) – основанную на ADO технологию, обеспечивающую удаленный доступ к источникам данных ADO с целью построения многоуровневых систем.

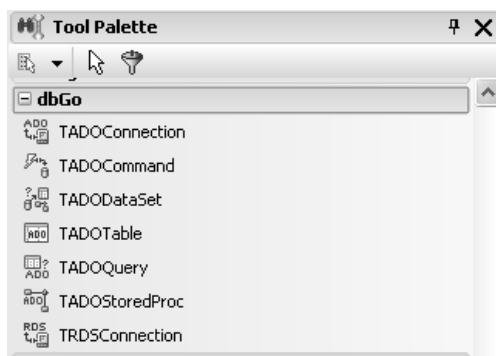


Рис. 3. Невизуальные компоненты для работы с ADO

С развитием технологии .NET всё большую популярность набирают технологии, созданные специально для .NET-приложений баз данных. Это в первую очередь технология ADO.NET от Microsoft и технология dbExpress.NET от Borland/CodeGear. Однако, поскольку в данном цикле лабораторных работ рассматривается разработка классических Windows-приложений, .NET-технологии не затрагиваются.

3. ОБЗОР НЕВИЗУАЛЬНЫХ КОМПОНЕНТОВ

3.1. ADOConnection

Главным невидуальным компонентом ADO, с которого «начинается» приложение баз данных, является ADOConnection. Этот компонент позволяет настраивать параметры подключения к конкретной БД (более точно, к конкретному источнику данных). Поэтому ADOConnection является «точкой подключения» для остальных невидуальных компонентов.

Поместите ADOConnection на форму и настройте основное свойство ConnectionString (строка соединения). При щелчке на кнопке с многоточием вызывается форма выбора источника строки соединения (рис. 4).

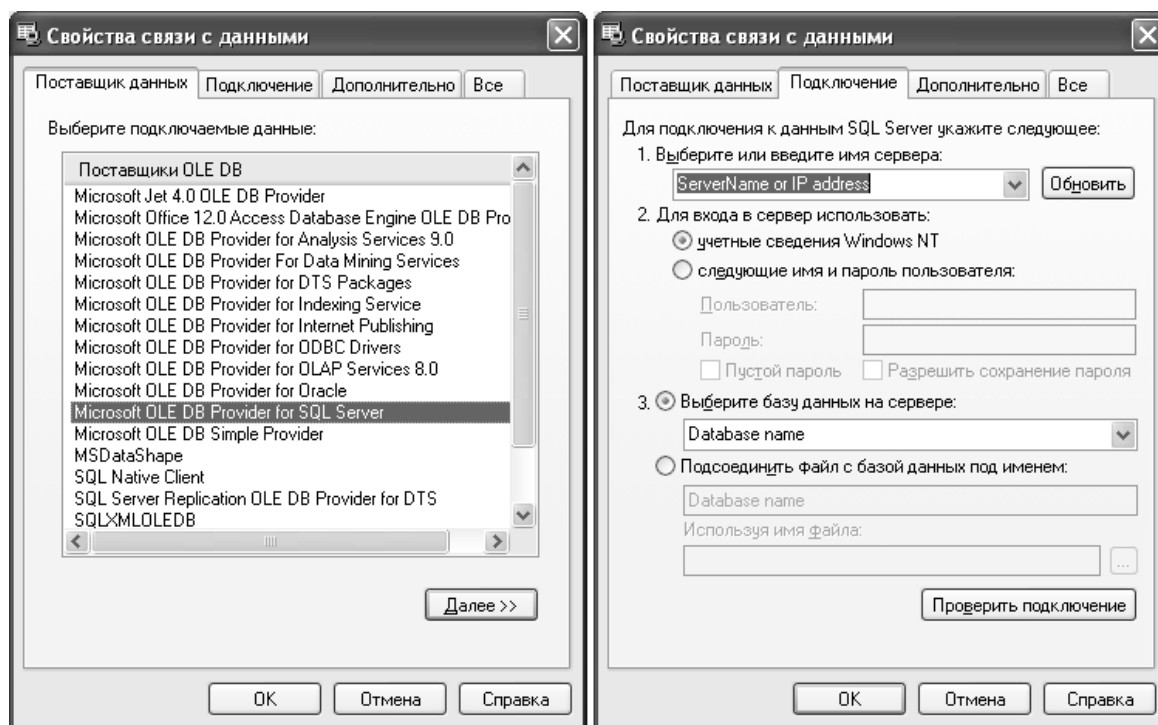
Вариант Use Data Link File позволяет указать в качестве источника строки соединения внешний файл (с расширением .udl). Это вариант наиболее предпочтителен, так как позволяет менять параметры соединения, не меняя бинарный код приложения. Вариант Use Connection String ука-

зывает на то, что строка соединения будет задана непосредственно и храниться в теле программы.



Рис. 4. Настройка источника строки соединения

Щёлкните на кнопке Build. Появится интерактивный редактор строки соединения. На первой странице Поставщик данных (Data Provider) следует выбрать поставщика данных, как показано на рис. . 5, а. Для работы с СУБД Microsoft SQL Server выберите Microsoft OLE DB Provider for SQL Server.



а

б

Рис. 5. Настройка параметров соединения

Далее следует перейти на страницу Подключение, которая зависит от выбранного поставщика. Вид страницы для указанного в примере поставщика показан на рис. . 5, б. Следует задать имя SQL-сервера или его IP-адрес. Для сервера, работающего на данном компьютере, указать

(local). Затем выбрать тип аутентификации: аутентификация Windows или аутентификация SQL-сервера. Затем следует задать имя базы данных.

Этих параметров достаточно, можно завершить настройку и вернуться в среду разработки.

Если используется аутентификация Windows, то свойству LoginPrompt (показывать диалог ввода пароля) следует присвоить False.

Для того чтобы начать сеанс работы с БД следует присвоить свойству Connected значение True. При этом будет выполнена попытка установить соединение с указанным сервером в указанном режиме аутентификации для указанной БД.

Если соединение произошло успешно, свойство Connected имеет значение True, в случае ошибок и проблем – False.

Для того чтобы завершить сеанс работы с БД следует присвоить свойству Connected значение False.

Внимание, совет! Всегда выполняйте подключение к БД только программно, после загрузки приложения. В режиме design-time присваивайте Connected значение True только временно, в целях настройки и проверки, после чего не забывайте сбросить его в False.

Если требуется создать UDL-файл, предназначенный для хранения строки описания соединения, которая будет многократно использоваться в дальнейшем (в варианте Use Data Link File), то можно воспользоваться следующим алгоритмом. Сначала любым способом создать пустой файл с расширением .udl в проводнике Windows. Затем двойным щелчком на этом файле вызвать редактор строки соединения, описанный выше.

3.2. Соединение с вводом пароля

Для того чтобы подключиться к БД с указанием имени и пароля (например, в режиме аутентификация SQL-сервера), следует назначить обработчик события OnWillConnect вашего ADOConnection:

```
procedure TMainForm.ADOConnectionWillConnect(Connection: TADOConnection;  
    var ConnectionString, UserID, Password: WideString;  
    var ConnectOptions: TConnectOption; var EventStatus: TEventStatus);  
begin  
    UserID := PasswordForm.EditLogin.Text;  
    Password := PasswordForm.EditPassword.Text;  
end;
```

Это событие вызывается при попытке активизировать соединение (например, путём присвоения Connected значения True), но ещё до того, как будет выполнена попытка соединения. Обработчик позволяет присвоить значения UserID (имя пользователя, login) и Password (пароль),

передаваемые серверу. В указанном примере серверу передаются значения, введённые пользователем в поля ввода EditLogin и EditPassword типа TEdit, которые находятся на форме авторизации PasswordForm.

Следовательно, необходимо перед этим создать форму авторизации PasswordForm, однако эта форма не должна быть главной. Вызывать эту форму лучше всего из обработчика события OnShow главной формы:

```
procedure TMainForm.FormShow(Sender: TObject);
begin
  if PasswordForm.ShowModal() <> mrOK then
    Close();
end;
```

На форме авторизации кнопке «ОК» следует назначить такой обработчик события OnClick:

```
procedure TPasswordForm.ButtonOKClick(Sender: TObject);
begin
  MainForm.ADOConnection.Connected := True;
  if not MainForm.ADOConnection.Connected then
    Exit;
  ModalResult := mrOK;
end;
```

3.3. ADOQuery

Это главный невизуальный компонент, с которым программисту приходится иметь дело. Этот компонент представляет собой набор данных (dataset), извлекаемых из БД по запросу.

Текст SQL-запроса задаётся и редактируется через свойство SQL компонента TADOQuery. Для запросов к Microsoft SQL Server следует писать запросы на Transact-SQL, используя любые желаемые возможности этого языка.

Для связи компонента ADOQuery с соединением ADOConnection служит свойство Connection.

Чтобы выполнить («активировать») запрос, следует присвоить свойству Active значение True. В качестве альтернативы можно вызвать метод Open(), который, впрочем, делает то же самое.

Если закрыть запрос, то есть присвоить свойству Active значение False или вызвать метод Close(), все данные из памяти компонента будут очищены.

Внимание, совет! Всегда выполняйте активацию запроса только программно, причём в тот момент, когда эти данные действительно понадобятся. В режиме design-time присваивайте Active значение True только временно, в целях настройки и проверки, после чего не забывайте сбросить его в False.

После выполнения запроса СУБД возвращает набор записей (recordset), который хранится во внутренней памяти компонента. Компонент предоставляет методы и свойства для перемещения по набору записей, обращения к значениям полей, редактирования данных (вставки, удаления и изменения записей) и много другого.

Чтобы понять, как правильно работать с этим компонентом, надо представить его как массив записей, причем в каждый момент времени одна, и только одна из них является активной. Все операции чтения и записи значений полей всегда относятся только к текущей, активной записи.

3.4. Навигация по набору данных

Методы First() и Last() перемещают указатель текущей записи к первой и последней записям в наборе данных соответственно, а методы Next() и Prior() – на одну запись вперед или назад соответственно. Ниже следующий пример показывает, как организовать перебор записей в запросе с помощью свойства EOF, показывающего «отсутствие следующей записи»:

```
QueryCustomer.First;           // Переход к началу набора данных
while not QueryCustomer.EOF do // Перебор всех записей в наборе
begin
    // Выполнение обработки текущей записи
    QueryCustomer.Next;         // Перемещение к следующей записи
end;
```

Кроме того, для навигации по набору данных иногда полезно использовать его свойства RecordCount и RecNo. Первое показывает количество записей в наборе данных, а второе – номер текущей записи во внутреннем массиве записей (номера считаются с нуля).

Используя номер текущей записи, следует помнить, что он может измениться при сортировке набора данных или при удалении и вставке записей.

3.5. Работа с полями

Delphi позволяет получить доступ к полям любого набора данных с помощью класса TField и его потомков. Класс TField позволяет не только считать или установить значение выбранного поля текущей записи набора данных, но и изменить характеристики поля посредством модификации его свойств. Кроме того, можно модифицировать набор данных в целом, изменяя визуальный порядок расположения полей, удаляя поля или же создавая новые вычисляемые или подстановочные поля.

Получить доступ к значениям полей записи очень просто. Компонент TADOQuery по умолчанию предлагает массив свойств с именем FieldValues[], который возвращает значение определенного поля как значение типа Variant. Поскольку массив FieldValues[] это массив свойств по умолчанию, вам не нужно определять имя свойства для доступа к массиву. Например, в следующем фрагменте кода значение поля CustName запроса Query1 присваивается переменной s типа String:

```
s := Query1[ 'CustName' ];
```

Получить доступ к объекту TField можно с помощью функции FieldByName():

```
var  
  CustNameField: TField;  
  s: string;  
begin  
  CustNameField := Query1.FieldByName('CustName');  
  s := CustNameField.AsString;
```

Существует возможность работы с полями набора данных во время проектирования (design-time). Чаще всего это необходимо, чтобы создать специальные поля-переменные (persistent field components), которые постоянно представляют непосредственный доступ к полям набора данных.

Правой кнопкой мыши щёлкните по компоненту набора данных (чаще всего ADOQuery). Выберите пункт Fields Editor. Откроется окно редактора полей.

Создание полей-переменных. В контекстном меню окна редактора полей выберите команду Add fields и добавьте все нужные поля (рис. 6).

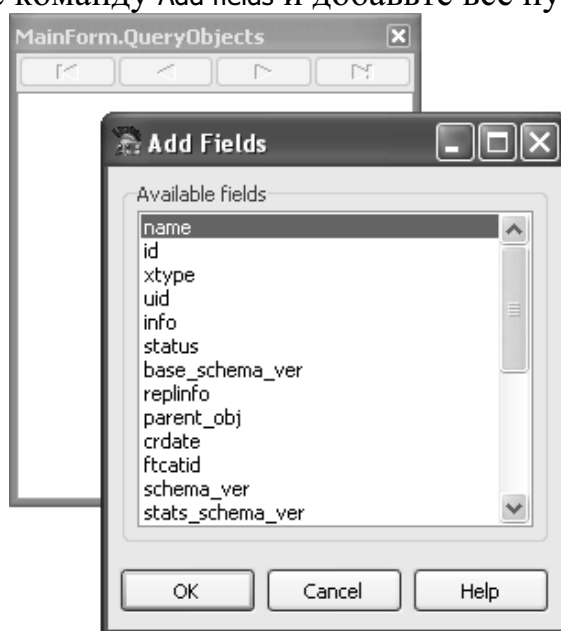


Рис. 6. Добавление полей-переменных

В результате для каждого выбранного поля набора данных в классе формы будет создано постоянное поле-переменная, которое можно снабдить нужным именем в «Инспекторе объектов» (Object Inspector).

Внимание! Для того чтобы получить список доступных полей набора данных среда Delphi пытается активировать его специальным образом. Если параметры соединения настроены неверно или в запросе есть ошибки, то при попытке выполнить команду Add fields вы получите сообщение об ошибке.

Созданные поля-переменные являются объектами одного из потомков класса TField, в соответствии с определенным типом поля в БД, например: TStringField для строкового поля, TIntegerField для целочисленного поля, TDateField для поля даты-времени, TFloatField для вещественного поля и т. д.

Например, если для поля «Name» запроса QueryObjects создано поле-переменная с именем QueryObjectsName, то доступ к его значению в текущей записи можно получить такими способами:

```
s := QueryObjects['Name']; //Способ 1  
s := QueryObjects.FieldName('Name').AsString; //Способ 1  
s := QueryObjectsName.AsString; //Способ 2, через поле-переменную
```

Доступ с помощью полей-переменных является наиболее быстрым и устойчивым к переименованию полей. При переименовании поля в наборе данных нужно будет только скорректировать свойство FieldName в соответствующем поле-переменной. Но главное, разумеется, это возможность настраивать различные параметры доступа к полю и назначать обработчики различных событий.

Работа с вычисляемыми полями. Помимо прочего, в окне редактора полей к набору данных можно добавить вычисляемые поля. Например, допустим, что в набор данных необходимо добавить поле, отображающее для каждой строки в запросе Orders объем оптовой продажи, составляющий 32 % от общего объема.

Выберите в контекстном меню окна редактора полей команду New Field. На экране раскроется диалоговое окно New Field, показанное на рис. 7. В поле Name этого окна введите имя нового поля WholeSaleTotal. Тип этого поля Currency, поэтому в раскрывающемся списке Type выберите именно это значение. В группе Field Type установите переключатель в положение Calculated и щелкните на кнопке ОК.

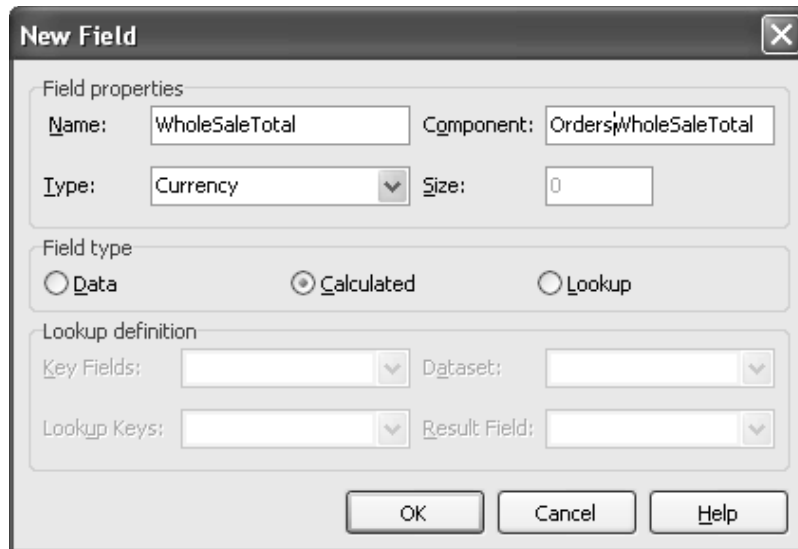


Рис. 7. Добавление вычисляемого поля

Чтобы заполнить новое поле данными, необходимо назначить требуемый метод событию OnCalcFields запроса Orders. В тексте обработчика этого события полю WholeSaleTotal следует просто присвоить значение, равное 32 % от существующего значения поля SalesTotal. Соответствующий текст метода обработки события Orders.OnCalcFields показан ниже:

```
procedure TMainForm.OrdersCalcFields( DataSet: TDataSet );
begin
    WholeSaleTotal := SalesTotal.AsFloat * 0.32;
end;
```

Работа с lookup-полями. Подстановочные (*lookup*) поля позволяют создавать в наборе данных такие поля, значения которых будут выбираться из другого набора данных. Для иллюстрации сказанного добавим такое поле к запросу Orders. Вряд ли по номеру клиента (поле CustNo таблицы Orders) можно будет вспомнить его имя. Поэтому целесообразно добавить к набору данных подстановочное поле, связанное с таблицей Customer, из которой по номеру клиента будет выбираться его имя.

Вначале поместите на форму второй набор данных Customer. Затем в запросе Orders вновь откройте окно New Field, для чего выберите команду New Field в контекстном меню окна редактора поля. Присвойте новому полю имя CustName, тип String, размер поля – 15. Не забудьте установить переключатель в группе Field Type в положение Lookup. В списке Dataset этого диалогового окна выберите значение Customer (именно этот набор данных необходимо просматривать). В обоих списках Key Fields и Lookup Keys диалогового окна выберите значение CustNo – это именно то общее поле, по значению которого будет выполняться поиск. И, наконец, в списке Result выберите значение FullName – именно это поле необходимо отображать в нашем наборе данных.

Работа с BLOB-полями. Поля *BLOB* (*Binary Large Object*) разработаны для размещения в них больших бинарных объектов, обычно файлов. *BLOB*-поля в одной записи набора данных могут содержать три байта данных, в то время как подобное поле в другой записи может содержать 3 мегабайта данных.

В библиотеке VCL существует класс `TBlobField`, производный от класса `TField` и специально предназначенный для работы с *BLOB*-полями. Главным образом на практике используются методы `SaveToFile()` и `LoadFromFile()` класса `TBlobField`.

Например, код выгрузки в файл с именем 'C:\temp\log.dat' содержимого *BLOB*-поля текущей записи 'Файл' выглядит следующим образом:

```
TBlobField(Query.FieldByName('Файл')).SaveToFile('C:\temp\log.dat');
```

3.6. Модификация данных

Условия редактируемости. Первое, что следует понять для модификаций в наборе данных, это те условия, при которых набор записей запроса вообще разрешается редактировать. Укажем некоторые (основные) условия:

1. SQL-запрос должен быть написан таким образом, чтобы в области FROM была указана только одна таблица. Самый простой пример:

```
SELECT * FROM Заявки WHERE [Дата заявки] > '01.01.2007'
```

2. SQL-запрос должен быть написан таким образом, чтобы из списка столбцов области SELECT не был удалён первичный ключ, то есть каждая запись должна чётко идентифицироваться.

3. У пользователя БД должны быть соответствующие права.

4. Свойство `CanModify` набора данных должно иметь значение `True` (определяется после выполнения запроса автоматически, может быть изменено).

Удаление записи. Метод `Delete()` удаляет из БД текущую запись:

```
Query.Delete();
```

Если при удалении произошли ошибки, запись не удаляется.

Обработчики событий `OnBeforeDelete` и `OnAfterDelete` позволяют выполнить дополнительную обработку соответственно до и после удаления. Событие `OnBeforeDelete` чаще всего используют для выдачи предупреждающих сообщений и для выполнения дополнительных проверок допустимости удаления.

Изменение записи. При попытке сразу изменить значение какого-нибудь поля текущей записи будет выдано сообщение об ошибке, поскольку набор данных сначала следует явно перевести в состояние редактирования вызовом метода `Edit()`.

После того, как набор данных переведён в состояние редактирования, можно менять значения полей:

```
with QueryCustomer do
begin
  Edit();
  FieldByName('Код').AsInteger := 1234;
  FieldByName('Имя').AsString := 'Bill Gates';
  FieldByName('Дата').AsDateTime := Now();
end;
```

Запись изменений в БД. При изменении значений полей текущей записи новые значения хранятся в памяти и не записываются в БД сразу. Для того чтобы записать все изменения в текущей записи в БД следует явно вызвать метод `Post()`:

```
procedure TEditCustomerForm.ButtonPostClick(Sender: TObject);
begin
  try
    QueryCustomer.Post();
  except on E: Exception do //При ошибке записи в БД
    begin
      Application.MessageBox( //выдать сообщение
        'Ошибка записи в БД: '#10 + E.Message,
        'Ошибка',
        MB_OK + MB_ICONERROR
      );
    end;
  end;
end;
```

Внимание! При перемещении на другую запись набора данных метод `Post()` вызывается автоматически, если в данных текущей записи сделаны какие-то модификации.

Если же требуется отменить все изменения, которые сделаны в полях текущей записи, но ещё не записаны в БД, следует вызвать метод `Cancel()`. Метод отменяет режим редактирования и переводит набор данных в состояние, в котором он был до внесения изменений.

Обработчики событий `OnBeforePost` и `OnAfterPost` позволяют выполнить дополнительную обработку соответственно до и после записи в БД. Событие `OnBeforePost` чаще всего используют для выполнения дополнительных проверок допустимости записи. Например, в нём можно проверять, все ли обязательные поля заполнены:

```

procedure TEditCustomerForm.CustomerBeforePost(DataSet: TDataSet);
begin
  if QueryCustomer.FieldName('Дата').IsNull then //Нет значения
  begin
    Application.MessageBox(
      'Не задано значение даты',
      'Ошибка',
      MB_OK + MB_ICONERROR
    );
    Abort(); //Отменить текущую операцию, в данном случае - Post()
  end;
end;
end;

```

Вставка записи. Вставка новой записи схематично происходит по следующему алгоритму:

1. Вставить в памяти набора данных новую пустую запись.
2. Заполнить поля новой (пустой) записи данными.
3. Сохранить запись в БД, либо отменить вставку.

Для вставки следует вызвать метод `Insert()` или `Append()`. Вставка новой (пустой) записи происходит в оперативной памяти компьютера и до явной записи в БД никак не затрагивают истинное содержимое БД.

Редактирование значений полей временной (вставленной) записи и запись её в БД можно выполнять теми же средствами, как и при редактировании существующей, например:

```

with QueryCustomer do
begin
  Insert();
  FieldByName('Код').AsInteger := 1235;
  FieldByName('Имя').AsString := 'Paul Allen';
  FieldByName('Дата').AsDateTime := Now();
  Post();
end;

```

Запись вставленной записи в БД осуществляется методом `Post()`, а отмена вставки – методом `Cancel()`.

Обработчики событий `OnBeforeInsert` и `OnAfterInsert` позволяют выполнить дополнительную обработку соответственно до и после вставки пустой записи в память набора данных. Событие `OnAfterInsert` или аналогичное событие `OnNewRecord` чаще всего используют для заполнения пустых полей новой записи значениями по умолчанию.

Внимание! При перемещении на другую запись набора данных метод `Post()` вызывается автоматически, если была вставлена запись.

3.7. Фильтрация набора данных

После того, как запрос выполнен, и набор данных заполнен записями, нередко возникает необходимость в дополнительном отборе записей из уже имеющегося в памяти набора. Это действие называется фильтрацией записей. Суть его состоит в том, что в результате применения дополнительного условия – фильтра – в наборе записей остаются «видимыми» только те записи, которые удовлетворяют условию. Остальные записи становятся «невидимыми», «недоступными».

Основное различие фильтрации по сравнению с явными условиями выборки WHERE в SQL-запросе состоит в том, последние обрабатываются СУБД в момент выполнения запроса, а фильтрация выполняется просто в памяти над уже открытым запросом, то есть не требует операций с БД.

Условие фильтрации задаётся в текстовом свойстве Filter как обычный предикат, например:

```
QueryCustomer.Filter := 'Код > 100';
```

Данное условие указывает оставить «видимыми», «доступными» только те записи, в которых поле «Код» имеет значение более 100.

Однако задание свойства Filter не приводит к немедленной фильтрации набора данных. Для применения фильтра следует установить свойство Filtered:

```
QueryCustomer.Filtered := True; //Применить фильтр
```

В фильтре можно использовать имена полей набора данных, строковые и численные константы, условия сравнения <, >, =, <=, >=, <>.

Отдельные подвыражения можно объединять логическими условиями OR или AND:

```
QueryCustomer.Filter := 'Код >= 100 AND Код <= 200';
```

Внимание! К сожалению, в условии фильтра можно использовать либо конъюнкцию, либо дизъюнкцию подвыражений, то есть можно использовать либо OR, либо AND.

В тексте фильтра строковые и символьные константы, значения даты и времени должны по правилам синтаксиса заключаться в апострофы (как и в языке SQL). Чтобы задать апостроф внутри строки в коде Delphi, следует удвоить его:

```
QueryCustomer.Filter := 'Имя = ''Иванов'' AND Дата > ''01.01.2008''';  
//Строка фильтра примет значение Имя = 'Иванов' AND Дата > '01.01.2008'
```

Другой способ «обрамления» апострофами состоит в использовании функции QuotedStr() из модуля SysUtils:

```
QueryObjects.Filter := 'Имя = ' + QuotedStr( EditCustomerName.Text );
```

При работе с СУБД Microsoft SQL Server и некоторыми другими СУБД можно задавать имена атрибутов, включающие пробелы. Для того, чтобы использовать такой атрибут в фильтре, его название следует заключить в квадратные скобки:

```
QueryStudents.Filter := '[Дата рождения] > ''01.01.1985''';
```

Если при включенной фильтрации вставить строку или отредактировать данные так, что строка не будет удовлетворять фильтру, то строку после записи строка «исчезнет» из видимого набора строк.

3.8. Состояние набора данных

Иногда бывает необходимо уточнить, в каком режиме находится набор данных (редактирования, вставки, просмотра и т. д.) и активен ли он вообще. Получить эту информацию можно с помощью свойства *State*. Свойство *State* имеет тип *TDataSetState*, а его значения приведены в табл. 1.

Таблица 1.

Возможные значения свойства State

Значение	Описание
<i>dsBrowse</i>	Обычный режим просмотра данных
<i>dsCalcFields</i>	Вызван обработчик события <i>OnCalcFields</i> , и значение полей записи в настоящий момент пересчитывается
<i>dsEdit</i>	Набор данных находится в режиме редактирования. Это означает, что был вызван метод <i>Edit()</i> , но отредактированная запись еще не внесена в БД
<i>dsInactive</i>	Набор данных закрыт
<i>dsInsert</i>	Набор данных находится в режиме <i>Insert</i> (вставка). Это означает, что был вызван метод <i>Insert ()</i> или <i>Append()</i> , но вставляемая запись еще не внесена в БД
<i>dsSetKey</i>	Набор данных находится в режиме <i>SetKey</i> (Задание ключа). Был вызван метод <i>SetKey()</i> , но метод <i>GotoKey()</i> еще не вызывался
<i>dsNewValue</i>	Набор данных находится во временном состоянии, когда осуществляется доступ к свойству <i>NewValue</i>
<i>dsOldValue</i>	Набор данных находится во временном состоянии, когда осуществляется доступ к свойству <i>OldValue</i>
<i>dsCurValue</i>	Набор данных находится во временном состоянии, когда осуществляется доступ к свойству <i>CurValue</i>
<i>dsFilter</i>	В наборе данных в настоящее время выполняется фильтрация записей с использованием события <i>OnFilterRecord</i>
<i>dsBlockRead</i>	Набор данных буферизируется, поэтому при установке этого значения перемещение курсора не приводит к обновлению данных в элементах управления и генерации событий
<i>dsInternalCalc</i>	В настоящее время вычисляется значение поля, у которого свойство <i>FieldKind</i> имеет значение <i>fkInternalCalc</i>
<i>dsOpening</i>	Набор данных находится в состоянии открытия, которое в настоящий момент еще не завершено. Это состояние устанавливается только тогда, когда набор данных открывается для асинхронной выборки данных

3.9. Поиск записи

Поиск данных в уже открытом наборе можно выполнить несколькими способами. Если необходимо найти несколько записей, удовлетворяющих некоторому критерию, следует использовать фильтрацию (см. п. 0).

Если необходимо найти конкретную запись по значениям определённых полей, следует использовать функцию `Locate()`. Если функция `Locate()` находит нужную строку, эта строка делается текущей, и возвращается значение `True`. Если поиск неудачен, функция возвращает `False`, и текущая запись не изменяется. Спецификацию функции следует изучать по справке, а здесь показаны примеры использования.

1. Найти строку, в которой поле «Код» имеет значение 12345:

```
Success := QueryCustomer.Locate('Код', 1235, [] );
```

2. Найти строку, в которой поле «Имя» имеет значение «bill gates» без учёта регистра, и поле «Дата» равно 01.01.2008:

```
Success := QueryCustomer.Locate(
  'Имя;Дата', VarArrayOf(['bill gates', '01.01.2008']),
  [loCaseInsensitive]
);
```

Наконец, если нет необходимости делать найденную запись текущей, а просто требуется по известным значениям одних полей найти неизвестные значения других полей, можно использовать функцию `Lookup()`. Спецификацию функции следует изучить по справке.

3.10. Модули данных

Иногда все невизуальные компоненты (соединения, запросы и т. д.) удобнее размещать прямо на той форме, на которой находится пользовательский интерфейс работы с этими данными.

Однако существуют причины, по которым в ряде случаев уместнее группировать невизуальные компоненты работы с БД на специальной форме, называемой модулем данных (*data module*). Модуль данных создаётся в Delphi командой `File > New > Other... > Delphi Files > Data Module`.

Модуль данных можно воспринимать как некоторую специальную форму-контейнер, которая видна только в режиме `design-time` и не предназначена для визуализации. На эту форму можно помещать только невизуальные компоненты.

Самая очевидная причина, по которой предпочтительнее использовать компонент `TDataModule`, нежели помещать компоненты доступа к данным на обычную форму, состоит в том, что это упрощает доступ к одним и тем же данным из нескольких форм и модулей проекта. Кроме того, в модуле данных можно сосредоточить весь код различных обра-

ботчиков событий невидимых компонентов, код инициализации и т. д. Всё это позволяет разгрузить основные формы, которые и без того зачастую очень и очень перенасыщены компонентами на самой форме и процедурами в теле модуля.

Более того, модулей данных можно создавать несколько, группируя, таким образом, наборы данных по некоторым критериям.

4. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

В принципе, создать клиентское приложение системы баз данных вполне можно на основе обычных компонентов просмотра и редактирования информации: TLabel, TEdit, TStringGrid, TMemo и т. п. в сочетании с использованием описанных выше методов и обработчиков событий невидимых компонентов БД.

Однако среда разработки Delphi вообще изначально создавалась специально с ориентацией на приложения баз данных. С этой целью были созданы специальные визуальные компоненты для просмотра и редактирования информации БД, которые делают ненужными значительную часть программирования. Более того, несложное приложение БД можно спроектировать и создать, вообще не прибегая к программированию.

Специальные компоненты ввода-вывода, которые можно связать с невидимыми компонентами БД, называют *data-aware controls*. Такие компоненты, которые входят в стандартную поставку Delphi, находятся на странице Data Controls палитры инструментов (рис. 8).

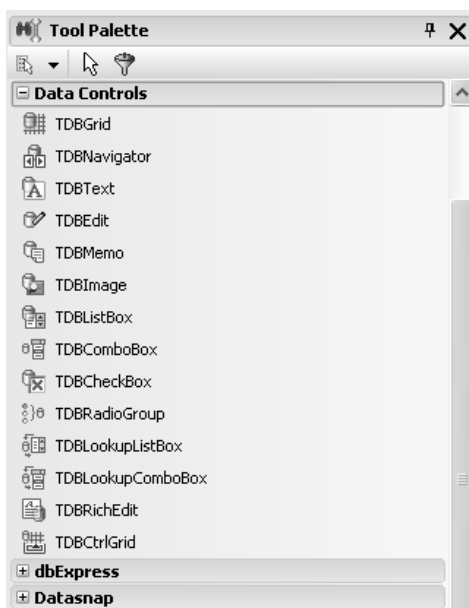


Рис. 8. Визуальные компоненты для работы с БД

4.1. Связывание с наборами данных

Сначала необходимо поместить визуальный компонент на форму. Затем, при просмотре его свойств, вы увидите свойство `DataSource` (источник данных). Источник данных – это невидимый компонент типа `TDataSource`, который является неким «буфером», связывающим визуальные компоненты с определенным набором данных, обычно – с запросом `ADOQuery`.

Итак, для каждого набора данных, который нужно связать с визуальными компонентами, надо создать объект типа `TDataSource`. Для этого в палитре инструментов следует открыть страницу `Data Access` (рис. 9), выделить компонент `TDataSource` и поместить его на форму или модуль данных.

У созданного объекта типа `TDataSource` имеется свойство `DataSet`, которому следует присвоить ссылку на необходимый набор данных.

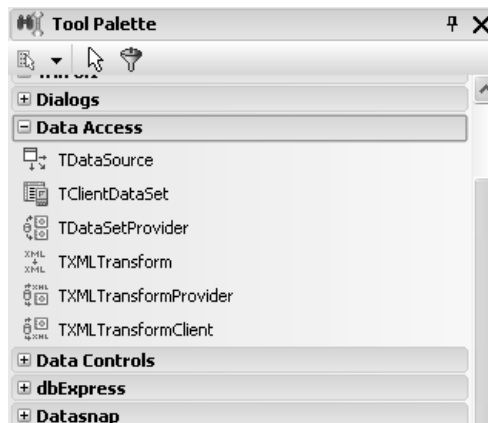


Рис. 9. Визуальные компоненты для работы с БД

Теперь можно связывать визуальные компоненты с нужным набором данных через созданный для него объект `DataSource`.

У объекта `DataSource` имеется свойство `AutoEdit`. Если это свойство имеет значение `True` (значение по умолчанию), то при попытке пользователя редактировать данные через любой из связанных визуальных компонентов, у набора данных автоматически вызывается метод `Edit()`, то есть набор данных переводится в состояние редактирования.

4.2. Визуальные компоненты

Визуальные компоненты для работы с БД можно условно разделить на два вида. Компоненты первого вида предназначены для отображения и редактирования одного конкретного поля текущей записи связанного набора данных. У таких компонентов помимо свойства `DataSource` необходимо задать свойство `DataField` – имя связанного поля.

Компоненты второго вида предназначены для отображения и редактирования связанного набора данных в табличном виде. На рис. 10 показан пример формы, на которой одновременно используются компоненты обоих видов. В верхней части находится компонент DBGrid, отображающий данные в табличном виде. В нижней части помещены компоненты типа DBEdit, которые отображают одиночные поля текущей записи.

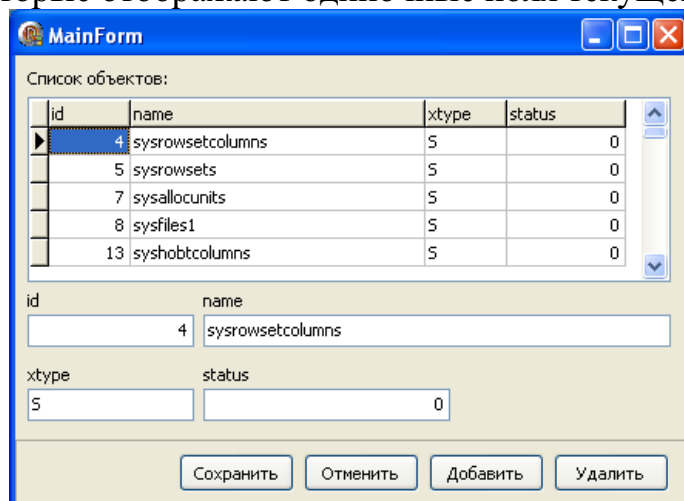


Рис. 10. Пример использования визуальных компонентов

Стандартные визуальные компоненты рассмотрены в табл. 2.

Таблица 2

Назначение визуальных компонентов

Компонент	Назначение
TDBText	Статически отображает текущее значение поля. Не предназначен для редактирования. Поддерживает поля простых типов: числа, строки, даты
TDBEdit	Data-aware-версия компонента TEdit. Отображает и редактирует текущее значение поля. Поддерживает поля простых типов: числа, строки, даты
TDBMemo	Data-aware-версия компонента TMemo. Отображает и редактирует текущее значение строкового поля, в том числе большого размера
TDBRichEdit	Data-aware-версия компонента TRichEdit. Отображает и редактирует текущее значение строкового поля, в том числе большого размера, в котором хранится форматированный текст формата RTF
TDBCheckBox	Data-aware-версия компонента TCheckBox. Отображает и редактирует текущее значение поля логического типа (True/False)
TDBImage	Data-aware-версия компонента TImage. Отображает и редактирует текущее значение BLOB-поля, хранящего изображение

Компонент	Назначение
TDBListBox	Отображает и редактирует текущее значение поля, принимающее значение из статического, заранее известного списка вариантов, например <i>пол</i> (мужской/женский), <i>цвет</i> (красный, оранжевый, ...) и т. д. Компонент отображает варианты в виде, типичном для <code>ListBox</code>
TDBComboBox	Отображает и редактирует текущее значение поля, принимающее значение из статического, заранее известного списка вариантов, подобно компоненту <code>TDBListBox</code> . Компонент отображает варианты в виде, типичном для <code>ComboBox</code>
TDBLookupListBox	Отображает и редактирует текущее значение поля, принимающее значение из списка вариантов, подобно <code>TDBListBox</code> . Однако список вариантов значений динамически формируется из другого набора данных, указанного в свойстве <code>ListSource</code> . Компонент отображает варианты в виде, типичном для <code>ListBox</code>
TDBLookupComboBox	Отображает и редактирует текущее значение поля, принимающее значение из списка вариантов, подобно <code>TDBComboBox</code> . Однако список вариантов значений динамически формируется из другого набора данных, указанного в свойстве <code>ListSource</code> . Компонент отображает варианты в виде, типичном для <code>ComboBox</code>
TDBGrid	Отображает записи набора данных в виде таблицы. Поля соответствуют столбцам, записи – строкам. <code>DBGrid</code> позволяет редактировать поля текущей записи и перемещаться по набору данных
TDBNavigator	Позволяет перемещаться по записям и вызывать операции редактирования набора данных: <code>Insert</code> , <code>Delete</code> , <code>Post</code> , <code>Cancel</code> и т. п.

Компонент `TDBLookupComboBox` идеален для редактирования полей, которые являются внешними ключами. Источник списка настраивается на главный набор данных, а редактируемое поле – на внешний ключ подчинённого набора данных.

Компонент `DBGrid` является наиболее популярным средством отображения и редактирования данных. Однако необходимо помнить, что табличный вид пригоден при условии того, что все столбцы видны одновременно, причём их ширина достаточна для полноценного просмотра и редактирования значений. Необходимость горизонтальной прокрутки и наличие длинных строковых полей резко снижают применимость компонента `DBGrid`. В этом случае его удобно комбинировать с компонентами первого вида, подобно тому, как это показано на рис. 10.

4.3. Синхронный просмотр

Очень часто встречается необходимость отображать данные сразу из двух таблиц, связанных внешним ключом. Например, главная таблица «Преподаватели» связана с подчинённой таблицей «Предметы» через целочисленное поле «Код преподавателя». При просмотре данных преподавателя было бы очень удобно отображать записи по тем предметам, которые он преподаёт. При перемещении по записям главного набора данных подчинённый набор данных должен синхронно «показывать» связанные записи. Такой способ просмотра называется синхронным.

Опишем один из способов организации синхронного просмотра. Для этого в главном наборе данных `QueryLectures` необходимо создать обработчик события `AfterScroll`, в котором выполняется фильтрация подчинённого набора данных `QuerySubjects`:

```
procedure TMainForm.QueryLecturesAfterScroll(DataSet: TDataSet);
begin
  if QueryLectures.State <> dsBrowse then
    Exit;
  QuerySubjects.Filter :=
    '[Код преподавателя] = ' +
    QuotedStr( QueryLectures.FieldByName('Код преподавателя').AsString );
  QuerySubjects.Filtered := True;
end;
```

4.4. Альтернативные способы отображения данных

Стандартные визуальные `data-aware`-компоненты предлагают, по сути, только два способа отображения данных: табличный и поэлементный. Другие способы требуют использования либо нестандартных компонентов от сторонних производителей, либо обычных не `data-aware`-компонентов на основе дополнительного программирования.

Один из наиболее популярных способов отображения данных типа главный/подчинённый (подобно описанному в п. 0) основан на использовании компонента `TTreeView`. Узлы верхнего уровня обычно соответствуют записям главного набора данных, а подчинённые узлы – связанным записям подчинённого набора. При выделении в дереве некоторого узла в информативной части формы обычно отображаются все необходимые атрибуты (рис. 11).

Программирование древовидного представления является достаточно непростым делом, требующим учёта и реализации многих операций. Однако получаемый результат обычно стоит затраченных усилий.

Древовидное представление является компактным, наглядным и обеспечивает удобную навигацию.

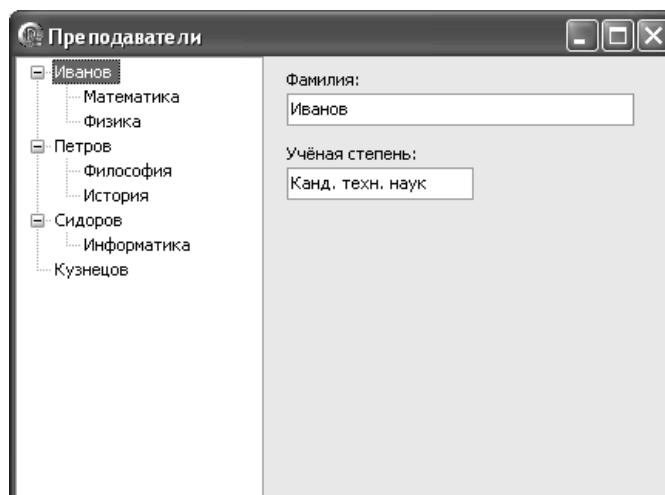


Рис. 11. Пример использования визуальных компонентов

Для организации такого интерфейса требуется следующее.

1. Реализация первичного построения дерева. Это самая несложная задача, поскольку дерево строится с помощью двух вложенных циклов. Внешний цикл организуется по главному набору данных. Пример кода перебора приведён в п. 0. Для каждой записи главного набора создаётся узел дерева верхнего уровня.

Подчинённые узлы создаются во время перебора подчинённых записей. Для отбора связанных подчинённых записей используется фильтрация, совершенно аналогично коду, приведённому в п. 0.

2. Обработка события выбора узла дерева. Для этого используется обработчик события `OnChange` компонента `TreeView`. Главная задача в обработчике – идентифицировать типа узла и найти в наборе данных запись, соответствующую данному узлу. После этого отображение данных этой записи является тривиальным.

Для решения этой задачи необходимо, чтобы к каждому узлу ещё при его создании была присоединена необходимая для идентификации информация. Наилучший способ – хранить в поле `Data` узла `TreeNode` ссылку на объект, содержащий необходимые поля (как минимум, тип узла и значение первичного ключа). Для этого, в свою очередь, необходимо заранее создать класс с нужными полями.

Если все предусловия выполнены, то в обработчике события `OnChange` из выделенного узла извлекается значение первичного ключа, по которому функцией `Locate()` выполняется позиционирование на соответствующей записи набора данных (см. п. 0).

3. Обработка вставки и удаления записей. Вставка и удаление записей в главном или подчинённом наборах данных, отображаемых в дереве, должны сопровождаться соответствующими добавлениями или уда-

лениями узлов в дереве. При удалении узла надо не забывать освобождать привязанный к нему через поле Data объект идентификации.

Мы рассмотрели древовидное представление с двумя уровнями. Очевидно, что при необходимости можно точно так же организовать работу с большим числом уровней иерархии.

Загружать сразу все данные в дерево следует при относительно небольших объёмах этой информации (не более сотен записей). В случае больших объёмов желательно загружать информацию о вложенных узлах только в момент раскрытия родительских узлов, если она не была ещё загружена.

5. НАПИСАНИЕ ЗАПРОСОВ

Простейшая структура оператора SELECT:

```
SELECT
  <список_полей>
FROM
  <табличное_выражение>
WHERE
  <критерий_выборки>
GROUP BY <список_полей_группировки>
HAVING <критерий_выборки_групп>
ORDER BY <список_полей_сортировки>
```

Ключевые слова SELECT и (как правило) FROM обязательны для SQL-оператора SELECT, все остальные ключевые слова – нет.

<Список_полей> в операторе SELECT – это список, состоящий из одного или более полей, которые будут включены в результирующий набор записей, возвращаемых оператором. Поля могут быть взяты из таблиц либо вычислены. <Табличное_выражение> в выражении FROM – это, чаще всего, имя таблицы или представления (View), либо соединение нескольких таблиц/представлений.

Ключевое слово WHERE предвывает предикат <критерий_выборки>, ограничивающий записи, которые будут включены в результирующий набор.

Оператор GROUP BY объединяет в одну те записи, в которых указанные поля <списка_полей_группировки> имеют одинаковые значения. Ключевое слово HAVING используется для задания дополнительных критериев отбора групп, полученных в результате выполнения оператора GROUP BY.

Оператор ORDER BY сортирует набор записей в том порядке, в котором они указаны в <списке_полей_сортировки>.

Пример 1. Выборка всех столбцов и всех строк

```
SELECT
    staffNo, fName, IName, position, sex, DOB, salary, branchNo
FROM
    Staff
```

Поскольку выборка всех имеющихся в таблице столбцов выполняется достаточно часто, в языке SQL определен упрощенный вариант записи значения «все столбцы» – вместо имен столбцов указывается символ звездочки (*). Приведенный ниже оператор полностью эквивалентен первому и представляет собой упрощенный вариант записи того же самого запроса:

```
SELECT
    *
FROM
    Staff
```

Пример 2. Выборка конкретных столбцов и всех строк

```
SELECT
    staffNo, fName, IName, Salary
FROM
    Staff
```

В этом примере на основе таблицы **Staff** создается новая таблица, включающая только указанные в запросе столбцы **staffNo**, **fName**, **IName** и **salary**.

Пример 3. Удаление дубликатов

Необходимо вывести список всех уникальных значений столбца **Salary**. Для того чтобы исключить из результата возможные дубликаты записей следует указать ключевое слово **DISTINCT**:

```
SELECT DISTINCT
    Salary
FROM
    Staff
```

Пример 4. Вычисляемые поля

```
SELECT
    [Название товара], Цена, Количество, (Цена*Количество) AS
    [Стоимость]
FROM
    [Покупки]
```

В этом примере в результирующем отношении появится новый атрибут «Стоимость», который является результатом расчётов заданного выражения для каждой исходной записи.

Пример 5. Простые условия выборки

Перечислите весь персонал с размером зарплаты больше 10000:

```
SELECT
    staffNo, fName, IName, Salary
FROM
    Staff
WHERE
    Salary > 10000
```

Более сложные условия могут быть объединениями подвыражений с использованием логических условий:

```
SELECT
    *
FROM
    Branch
WHERE
    City = 'London' OR City = 'Glasgow'
```

Альтернативная запись предыдущего условия:

```
WHERE
    City IN ('London', 'Glasgow')
```

Такая запись очень удобна при большом количестве вариантов.

Если же необходимо выбрать все значения, кроме некоторых, можно использовать либо

```
City <> 'London' AND City <> 'Glasgow'
```

либо NOT IN:

```
City NOT IN ('London', 'Glasgow')
```

Для определения принадлежности к интервалам значений можно использовать BETWEEN/NOT BETWEEN:

```
WHERE
    Salary BETWEEN 20000 AND 30000
```

Границы, указанные в BETWEEN, считаются принадлежащими интервалу.

Значение NULL в условиях выборки напрямую использовать нельзя, так как сравнение с NULL всегда даёт NULL. Следует использовать специальные операторы IS NULL/IS NOT NULL:

```
WHERE
    Comment IS NOT NULL
```

Пример 6. Условия выборки с указанием шаблонов

Для поиска строк по подстроке используются операторы LIKE/NOT LIKE. Например: найти все товары, в названии которых есть слово «сыр»:

```
SELECT  
    [Название товара]  
FROM  
    Товары  
WHERE  
    [Название товара] LIKE ' %сыр %'
```

Знак процента (%) означает любую подстроку. Знак подчёркивания (_) означает любой одиночный символ.

ЗАКЛЮЧЕНИЕ

Предыдущая часть пособия была посвящена средства проектирования баз данных и СУБД Microsoft SQL Server 2005.

Данная часть является логическим продолжением первой части. Методическое пособие полностью посвящено современным средствам и технологиям, на которых строится практическая часть дисциплины, а именно средствам разработки приложений баз данных на примере Borland/CodeGear Delphi 2007.

Объем пособия мал для полноценного освоения рассмотренных продуктов, но для «быстрого старта» он достаточен. В дальнейшем рекомендуется постоянно обращаться к электронной справке и дополнительной литературе.

ОГЛАВЛЕНИЕ

Введение	3
1. Архитектура системы баз данных	4
2. Архитектура компонентов баз данных VCL.....	6
3. Обзор невизуальных компонентов	8
3.1. ADOConnection.....	8
3.2. Соединение с вводом пароля	10
3.3. ADOQuery	11
3.4. Навигация по набору данных.....	12
3.5. Работа с полями.....	12
3.6. Модификация данных.....	16
3.7. Фильтрация набора данных	19
3.8. Состояние набора данных	20
3.9. Поиск записи.....	21
3.10. Модули данных	21
4. Создание пользовательского интерфейса	22
4.1. Связывание с наборами данных	23
4.2. Визуальные компоненты	23
4.3. Синхронный просмотр.....	26
4.4. Альтернативные способы отображения данных.....	26
5. Написание запросов	28
Заключение.....	31

Учебное издание

МИРОШНИЧЕНКО Евгений Александрович
ШЕСТАКОВ Николай Александрович

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические указания к циклу лабораторных работ
по дисциплине «Проектирование информационных систем»
для магистрантов, обучающихся по магистерской программе
«Компьютерный анализ и интерпретация данных»
направления 230100 «Информатика и вычислительная техника»

Научный редактор
доктор технических наук,
профессор

В.Г. Спицын

Верстка
Дизайн обложки

*В.П. Аршинова
О.Ю. Аршинова
О.А. Дмитриев*

Подписано к печати 25.11.2008. Формат 60x84/16. Бумага «Снегурочка».


Печать XEROX. Усл. печ. л. 1,92. Уч.-изд. л. 1,74.

Заказ 849. Тираж 100 экз.



Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 001:2000



ИЗДАТЕЛЬСТВО  ТПУ. 634050, г. Томск, пр. Ленина, 30.