**Жукова Т.А.**

# Разработка систем управления базами данных средствами Delphi

**учебное пособие**

**Шымкент, 2019**

УДК 004.7
ББК 32.973.202я7
Ж 86

Утверждено к печати учебно – методическим советом Университета «Мирас» (протокол № 7 от 27 февраля 2019 г.

Рецензенты:
к.т.н., зав.каф. Кошкинбаева М.Ж. – Университет «Мирас»;
к.т.н., ст. преподаватель Дуйсенов Н.Ж. – Университет «Мирас».

Жукова Т.А. Интернет – технологии. Учебное пособие - Шымкент: Университет «Мирас», 2018. – 238 с.

ISBN 978-9965-32-668-4

В учебно-методическом пособии, представлены теоретические сведения, способствующие приобретению знаний, умений и навыков студентами, связанных с приобретением знаний о современных методах и средствах разработки СУБД, изучением теоретической базы для усвоения специальных дисциплин, умением применять полученные знания в процессе учебы и на практике, приобретением практических навыков по разработке СУБД, проектирования и управления разработанной программой.

.

Данное пособие имеет своей целью -дать студентам систематизированные сведения о создании систем управления базами данных в среде объектно-ориентированного программирования; формирование знаний и умений в области создания и разработки систем управления базами данных средствами объектно-ориентированного программирования; приобретение знаний об основных теоретических и методических принципах разработки системы управления базами данных средствами Delphi, создания запросов и отчетов; умение обосновано использовать на практике и в процессе учебы разработанные СУБД; овладение теоретической базой для усвоения других специальных дисциплин, связанных с изучением систем управления базами данных, проектированием сложных информационных процессов.

Учебное пособие предназначено для преподавателей и студентов высших учебных заведений, изучающих курс «Разработка СУБД средствами Delphi».

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Любая организация нуждается в своевременном доступе к информации. Ценность информации в современном мире очень высока. Роль распорядителей информации чаще всего выполняют базы данных. Базы данных обеспечивают надежное хранение информации, в структурированном виде и своевременный доступ к ней. Практически любая современная организация нуждается в базе данных, удовлетворяющей те или иные потребности по хранению, управлению и администрированию данных.

Использование баз данных и информационных систем становится неотъемлемой составляющей деловой деятельности современного человека и функционирования организаций.

Разработка информационных систем становится широко распространенной задачей, решаемой различными государственными органами управления и коммерческими организациями. Поэтому важно разобраться, что собой представляют информационные системы, выделить наиболее характерные области их применения.

Информационные системы обеспечивают сбор, хранение, обработку, поиск, выдачу информации, необходимой в процессе принятия решений задач из любой области. Они помогают анализировать проблемы и создавать новые продукты. Информационная система - взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели. Существует много видов информационных систем: системы обработки данных, информационные системы управления, маркетинговые системы, системы бухгалтерского учета и другие, используемые в различных организациях. Огромное количество видов информационных систем породило большое число методологий и технологий их создания.

Актуальность данного учебного пособия заключается в том, что разработка приложений баз данных является одной из наиболее востребованных возможностей среды программирования Delphi, которая предоставляет разработчику большой набор простых в использовании инструментов, позволяющих быстро разрабатывать сложные проекты.

В процессе изучения данной дисциплины студенты должны получить следующие теоретические и практические навыки: проектирования базы данных и редактирования данных, создания и редактирования таблиц, знакомство со структурой полей, проектирования приложений для работы с БД, проектирования отчетов, использования запросов при проектировании приложений, проектирование меню, создания справочной системы.

## Тема № 1
### Архитектура приложений баз данных

**План:**

1. Как работает приложение баз данных
2. Модуль данных
3. Подключение набора данных
4. Настройка компонента TDataSource
5. Отображение данных

Приложение баз данных, как следует уже из его названия, предназначено для взаимодействия с некоторым источником данных — базой данных (БД). Взаимодействие подразумевает получение данных, их представление в определенном формате для просмотра пользователем, редактирование в соответствии с реализованными в программе бизнес - алгоритмами и возврат обработанных данных обратно в базу данных.

В качестве источника данных могут выступать как собственно базы данных, так и обычные файлы — текстовые, электронные таблицы и т. д. Но здесь мы будем рассматривать приложения, работающие с базами данных.

Как известно, базы данных обслуживаются специальными программами — системами управления базами данных (СУБД), которые делятся на локальные, преимущественно однопользовательские, предназначенные для настольных приложений, и серверные — сетевые (часто удаленные), многопользовательские, функционирующие на выделенных компьютерах — серверах. Главный критерий такой классификации — объем базы данных и средняя нагрузка на СУБД.

Тем не менее, несмотря на разнообразие реализаций, общая архитектура приложения баз данных остается неизменной.

Само приложение включает механизм получения и отправки данных, механизм внутреннего представления данных в том или ином виде, пользовательский интерфейс для отображения и редактирования данных, бизнес-логику для обработки данных.

Механизм получения и отправки данных обеспечивает соединение с источником данных (часто опосредованно). Он должен "знать", куда ему обращаться и какой протокол обмена использовать для обеспечения двунаправленного потока данных.

Механизм внутреннего представления данных является ядром приложения баз данных. Он обеспечивает хранение полученных данных в приложении и предоставляет их по запросу других частей приложения.

Пользовательский интерфейс обеспечивает просмотр и редактирование данных, а также управление данными и приложением в целом.

Бизнес-логика приложения представляет собой набор реализованных в программе алгоритмов обработки данных.

Между приложением и собственно базой данных находится специальное программное обеспечение (ПО), связывающее программу и источник данных и управляющее процессом обмена данными. Это ПО может быть реализовано самыми разнообразными способами, в зависимости от объема базы данных,

решаемых системой задач, числа пользователей, способами соединения приложения и базы данных. Промежуточное ПО может быть реализовано как окружение приложения, без которого оно вообще не будет работать, как набор драйверов и динамических библиотек, к которым обращается приложение, может быть интегрировано в само приложение. Наконец, это может быть отдельный удаленный сервер, обслуживающий тысячи приложений.

Источник данных представляет собой хранилище данных (саму базу данных) и СУБД, управляющую данными, обеспечивающую целостность и непротиворечивость данных.

В Delphi 7 реализовано достаточно большое число разнообразных технологий доступа к данным. Но последовательность операций при конструировании приложений баз данных остается почти одинаковой. И в работе используются по сути одни и те же компоненты, доработанные для применения с той или иной технологией доступа к данным.

В этой лекции рассматриваются общие подходы к разработке приложений баз данных в Delphi, базовые классы и механизмы, которые не изменятся, выберите ли вы для вашего приложения Borland Database Engine (BDE), Microsoft ActiveX Data Objects (ADO) или dbExpress.

В Репозитории Delphi отсутствует отдельный шаблон для приложения баз данных. Поэтому, как и любое другое приложение Delphi, приложение баз данных начинается с обычной формы. Безусловно, это оправданный подход, т. к. приложение баз данных имеет пользовательский интерфейс. И этот интерфейс создается с использованием стандартных и специализированных визуальных компонентов на обычных формах.

Визуальные компоненты отображения данных расположены на странице **Data Controls** Палитры компонентов. В большинстве они представляют собой модификации стандартных элементов управления, приспособленных для работы с набором данных.

Приложение может содержать произвольное число форм и использовать любой интерфейс (MDI или SDI). Обычно одна форма отвечает за выполнение группы однородных операций, объединенных общим назначением.

В основе любого приложения баз данных лежат наборы данных, которые представляют собой группы записей (их удобно представить в виде таблиц в памяти), переданных из базы данных в приложение для просмотра и редактирования. Каждый набор данных инкапсулирован в специальном компоненте доступа к данным. В VCL Delphi реализован набор базовых классов, поддерживающих функциональность наборов данных, и практически идентичные по составу наборы дочерних компонентов для технологий доступа к данным. Их общий предок — класс TDataSet.

Для обеспечения связи набора данных с визуальными компонентами отображения данных используется специальный компонент TDataSource. Его роль заключается в управлении потоками данных между набором данных и связанными с ним компонентами отображения данных. Этот компонент обеспечивает передачу данных в визуальные компоненты и возврат результатов редактирования в набор данных, отвечает за изменение состояния визуальных

компонентов при изменении состояния набора данных, передает сигналы управления от пользователя (визуальных компонентов) в набор данных. Компонент TDataSource расположен на странице **Data Access** Палитры компонентов.

Таким образом, базовый механизм доступа к данным создается триадой компонентов:

- компоненты, инкапсулирующие набор данных (потомки класса TDataSet);
- компоненты TDataSource;
- визуальные компоненты отображения данных.

Рассмотрим схему взаимодействия этих компонентов в приложении баз данных (рис. 1.1).

В приложении с источником данных или промежуточным программным обеспечением взаимодействует компонент доступа к данным, который инкапсулирует набор данных и обращается к функциям соответствующей технологии доступа к данным для выполнения различных операций. Компонент доступа к данным представляет собой "образ" таблицы базы данных в приложении. Общее число таких компонентов в приложении не ограничено.

С каждым компонентом доступа к данным может быть связан как минимум один компонент TDataSource. В его обязанности входит соединение набора данных с визуальными компонентами отображения данных. Компонент TDataSource обеспечивает передачу в эти компоненты текущих значений полей из набора данных и возврат в него сделанных изменений.

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

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

При открытии набора данных компонент обеспечивает передачу в набор данных записей из требуемой таблицы БД. Курсор набора данных устанавливается на первую запись. Компонент TDataSource организует передачу в компоненты отображения данных значений необходимых полей из текущей записи. При перемещении по записям набора данных текущие значения полей в компонентах отображения данных автоматически обновляются.

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

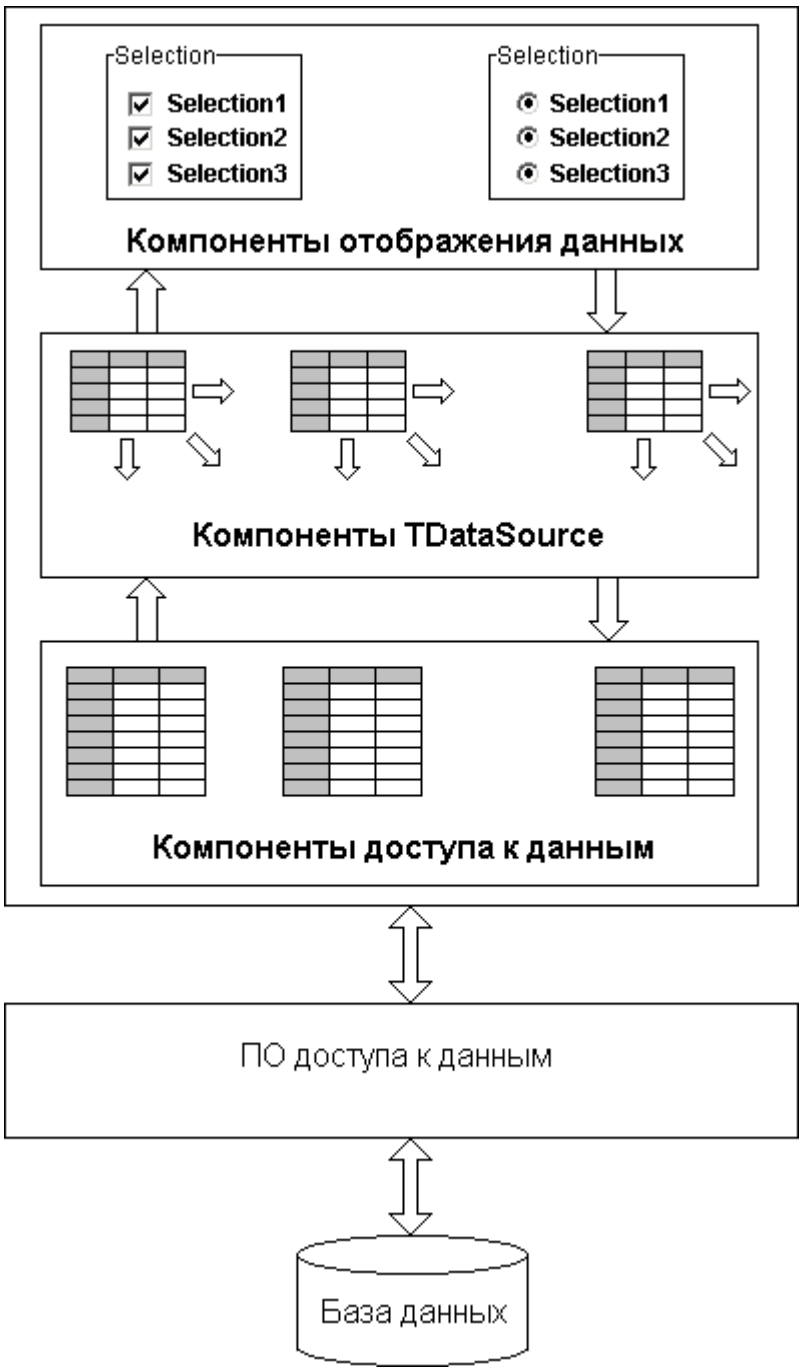


Рисунок 1.1. Механизм доступа к данным приложения баз данных

Теперь, имея общее представление о работе приложения баз данных, перейдем к поэтапному рассмотрению процесса создания такого приложения.

Модуль данных

Для размещения компонентов доступа к данным в приложении баз данных желательно использовать специальную "форму" - модуль данных (класс TDataModule). Обратите внимание, что модуль данных не имеет ничего общего с обычной формой приложения, ведь его непосредственным предком является класс TComponent. В модуле данных можно размещать только невизуальные компоненты. Модуль данных доступен разработчику, как и любой другой модуль проекта, на этапе разработки. Пользователь приложения не может увидеть модуль данных во время выполнения.

Для создания модуля данных можно воспользоваться Репозиторием объектов или главным меню Delphi. Значок модуля данных **Data Module** расположен на странице **New**.

Как уже говорилось, модуль данных имеет мало общего со стандартной формой, хотя бы потому, что класс TDataModule происходит непосредственно от класса TComponent. У него почти полностью отсутствуют свойства и методы-обработчики событий, ведь от платформы для других невизуальных компонентов почти ничего не требуется, хотя потомки модуля данных, работающие в распределенных приложениях, выполняют весьма важную работу.

Для создания структуры (модели, диаграммы) данных, с которой работает приложение, можно воспользоваться возможностями, предоставляемыми страницей **Diagram** Редактора кода. Любой элемент из иерархического дерева компонентов модуля данных можно перенести на страницу диаграммы и задать связи между ними.

При помощи управляющих кнопок можно задавать между элементами диаграммы отношения синхронного просмотра и главный/подчиненный. При этом производится автоматическая настройка свойств соответствующих компонентов.

Для создания модуля данных (рис. 1.2) можно воспользоваться Репозиторием объектов или главным меню Delphi. Значок модуля данных **Data Module** расположен на странице **New**.

Для обращения компонентов доступа к данным, расположенным в модуле данных, из других модулей проекта необходимо включить имя модуля в секцию uses:

```
unit InterfaceModule;
...
 implementation
uses DataModule;
...
DataModule.Tablel.Open;
...
```

Преимуществом размещения компонентов доступа к данным в модуле данных является то, что изменение значения любого свойства проявится сразу же во всех обычных модулях, к которым подключен этот модуль данных. Кроме этого, все обработчики событий этих компонентов, т. е. вся логика работы с данными приложения, собраны в одном месте, что тоже весьма удобно.

Подключение набора данных

Компонент доступа к данным является основой приложения баз данных. На основе выбранной таблицы БД он создает набор данных и позволяет эффективно управлять им. В процессе работы такой компонент тесно взаимодействует с функциями соответствующей технологии доступа к данным. Обычно доступ к функциональности технологии доступа к данным осуществляется через совокупность интерфейсов. Все компоненты доступа к данным являются невизуальными.

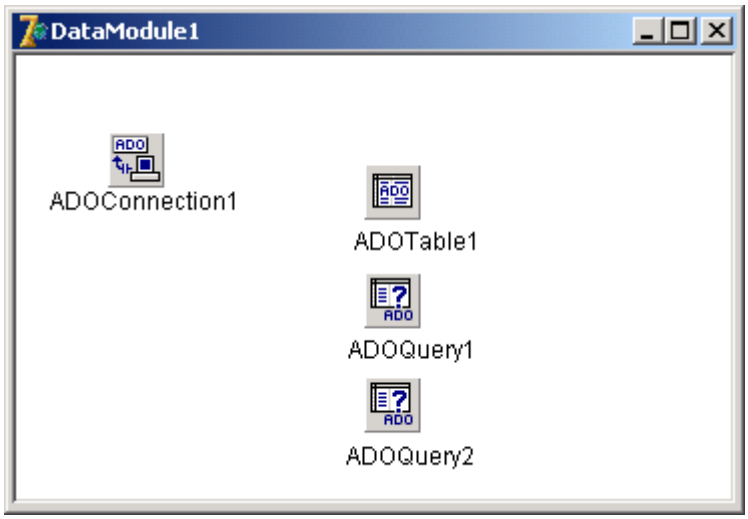


Рисунок 1.2. Модуль данных

Для создания нового проекта достаточно выбрать команду **New Application** из меню **File** или воспользоваться Репозиторием объектов, который открывается командой **New** из меню **File.**

**Примечание**

*Здесь рассматривается простейший вариант создания приложения. В реальных проектах для размещения компонентов доступа к данным следует использовать модуль данных.*

Затем на форму нового проекта необходимо перенести компонент, инкапсулирующий набор данных, и выполнить следующие действия. Последовательность действий рассмотрим для компонента, инкапсулирующего функции таблицы.

1. Подключить компонент к базе данных. Для этого, в зависимости от конкретной технологии, используется или специальный компонент, устанавливающий соединение, или прямое обращение к драйверу, интерфейсу или динамической библиотеке.

2. Подключить к компоненту таблицу БД. Для этого используется свойство TableName, доступное в Инспекторе объектов. После выполнения действий первого этапа в списке этого свойства должны появиться имена всех доступных в подключенной базе данных таблиц. После выбора имени таблицы в свойстве TableName компонент оказывается связанным с ней.

3. Переименовать компонент. Это не обязательное действие. Тем не менее, в любых случаях желательно присваивать компонентам доступа к данным осмысленные имена, соответствующие названиям подключенных таблиц. Обычно название компонента копирует название таблицы (например, Orders или OrdTable илт tblOrders).

4. Активизировать связь между компонентом и таблицей БД. Для этого используется свойство Active. Если в Инспекторе объектов присвоить этому свойству значение True, то связь активизируется. Эту операцию можно выполнить

и в исходном коде приложения. Также существует метод open, который открывает набор данных, и метод close, закрывающий его.

В качестве примера попробуем создать простейшее приложение баз данных, работающее с таблицей COUNTRY.DB из стандартной демонстрационной базы данных DBDEMOS через драйвер процессора Borland Database Engine.

На форму нового проекта необходимо перенести компонент TTable со страницы BDE Палитры компонентов. Свойство DatabaseName должно ссылаться на псевдоним DBDEMOS, который создается автоматически при установке Delphi, его можно выбрать из списка свойства DatabaseName. Для свойства TableName необходимо задать имя таблицы "COUNTRY.DB". Его также можно выбрать из списка. Двойной щелчок на свойстве Active в Инспекторе объектов присваивает ему значение True. После этого связь компонента с таблицей активизируется. Свойство Name имеет значение "CountryTable".

Открытие и закрытие набора данных можно предусмотреть как реакцию на действия пользователя или возникновение события. Чаще всего набор данных должен открываться при первом показе формы и закрываться при ее закрытии.

**Листинг 1.1.** Секция Implementation главного модуля проекта DemoDBApp

```
implementation
 {$R *.DFM}
procedure TForml.FormShowfSender: TObject);
begin
 try
CountryTable.Open;
 except
ShowMessage('Table open error');
 end;
end;
procedure TForml.FormClose(Sender: TObject; var Action: TCloseAction);
begin
CountryTable.Close;
end;
end.
```

При открытии формы выполняется метод обработчик FormShow. В нем набор данных открывается при помощи метода Open. Обратите внимание на использование конструкции try..except, которая обеспечивает корректное завершение при возникновении исключительных ситуаций.

Так как ошибки в работе приложений баз данных могут привести к серьезным последствиям (потеря или искажение данных), то защитный код должен присутствовать во всех критических местах.

В методе-обработчике FormClose, который вызывается при закрытии формы, набор данных закрывается методом close.

**Примечание**

*В принципе, для выполнения рассмотренных операций можно воспользоваться и свойством Active. Однако реальные операции выполняют указанные методы. Поэтому использование свойства является лишним этапом,*

*да и с точки зрения ООП все действия должны выполнять методы объекта, а свойства служат только для представления значений. Свойство Active сигнализирует о состоянии набора данных.*

Настройка компонента TDataSource

На втором этапе разработки приложения баз данных необходимо перенести на форму и настроить компонент TDataSource. Он обеспечивает взаимодействие набора данных с компонентами отображения данных. Чаще всего одному набору данных соответствует один компонент TDataSource, хотя их может быть несколько.

Для настройки свойств компонента необходимо выполнить следующие действия.

1. Связать набор данных и компонент TDataSource. Для этого используется свойство DataSet компонента TDataSource, доступное через Инспектор объектов. Это указатель на экземпляр компонента доступа к данным. В списке этого свойства в Инспекторе объектов перечислены все доступные компоненты наборов данных.

2. Переименовать компонент. Это не обязательное действие. Тем не менее желательно присваивать компонентам осмысленные имена, соответствующие названиям связанных наборов данных. Обычно название компонента комбинирует имя набора данных (например OrdSource или dsOrders).

В приложении DemoDBApp компонент countrysource связан с компонентом CountryTable. Поэтому свойство DataSet имеет значение CountryTable.

**Примечание**

*Компонент TDataSource можно подключить не только к набору данных из той же формы, но и любой другой, модуль которой указан в секции uses.*

Компонент TDataSource имеет ряд полезных свойств и методов. Итак, связывание с компонентом набора данных выполняет свойство

property DataSet: TDataSet;

а определить текущее состояние набора данных можно, используя свойство

type TDataSetState = (dslnactive, dsBrowse, dsEdit, dslnsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue, dsBlockRead, dsInternalCalc); property State: TDataSetState;

При помощи свойства

property Enabled: Boolean;

можно включить или отключить все связанные визуальные компоненты. При значении False ни один связанный компонент отображения данных не будет работать.

Свойство

property AutoEdit: Boolean;

при значении True всегда будет переводить набор данных в режим редактирования при получении фокуса одним из связанных визуальных компонентов.

Аналогично, метод

procedure Edit;

переводит связанный набор данных в режим редактирования.

Метод

function IsLinkedTo(DataSet: TDataSet): Boolean;

возвращает значение True, если компонент, указанный в параметре DataSet, действительно связан с данным компонентом TDataSource.

Метод-обработчик

type TDataChangeEvent = procedure(Sender: TObject; Field: TField)

of object;

property OnDataChange: TDataChangeEvent;

вызывается при редактировании данных в одном из связанных визуальных компонентов.

Метод-обработчик

property OnUpdateData: TNotifyEvent;

вызывается перед сохранением изменений в базе данных. Метод-обработчик

property OnStateChange: TNotifyEvent;

вызывается при изменении состояния связанного набора данных (см. гл. 12).

Отображение данных

На третьем этапе создания приложения баз данных необходимо разработать пользовательский интерфейс на основе компонентов отображения данных. Эти компоненты предназначены специально для решения задач просмотра и редактирования данных. Внешне большинство этих компонентов ничем не отличаются от стандартных элементов управления. Более того, многие из компонентов отображения данных являются наследниками стандартных компонентов — элементов управления.

Компоненты отображения данных должны быть связаны с компонентом TDataSource и через него с компонентом набора данных. Для этого используется их свойство DataSource. Оно присутствует во всех компонентах отображения данных.

Большинство компонентов предназначены для представления данных из одного единственного поля. В таких компонентах имеется еще одно свойство DataField, которое определяет поле связанного набора данных, отображаемое в компоненте.

Особое значение для приложений баз данных играет компонент TDBGrid, который представляет данные в виде таблицы. В столбцах таблицы размещаются поля набора данных, а в строках — записи. Для этого компонента не имеет смысла определять конкретное поле, но можно задать настраиваемый набор колонок, а для каждой из них определить поле набора данных.

Таким образом, для каждого визуального компонента отображения данных необходимо выполнить следующие операции:

1. Связать компонент отображения данных и компонент TDataSource. Для этого используется свойство Datasource, которое должно указывать на экземпляр требуемого компонента TDataSource. Один компонент отображения данных можно связать только с одним компонентом TDataSource. Необходимый компонент можно выбрать в списке свойств в Инспекторе объектов.

2. Задать поле данных. Для этого используется свойство DataField типа TFields. В нем необходимо указать имя поля связанного набора данных. После

задания свойства Datasource поле можно выбрать из списка. Этот этап применяется только для компонентов, отображающих единственное поле.

Отдельное место среди компонентов отображения данных занимает компонент TDBNavigator. Он предназначен для перемещения по записям набора данных.

В приложении DemoDBApp использованы компоненты TDBGrid, TDBNavigator и TDBEdit (рис. 1.3).

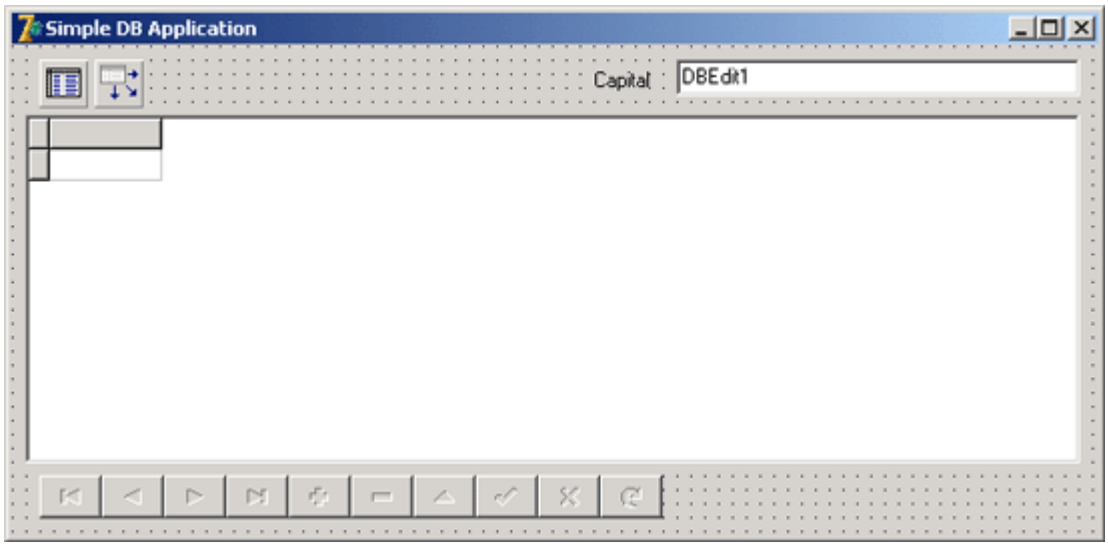


Рисунок 1.3. Главная форма приложения DemoDBApp

Все три компонента отображения данных связаны с компонентом CountrySource типа TDataSource при помощи свойства DataSource.

Компонент TDBEdit отображает данные из поля capital (столица государства) и позволяет редактировать их.

Компонент TDBGrid показывает набор данных целиком, данные в ячейках можно редактировать.

Компонент TDBNavigator позволяет перемещаться по записям набора данных CountryTable. При этом результат заметен во всех подключенных к набору данных компонентах отображения данных.

Резюме

Приложения баз данных могут получать доступ к источникам данных при помощи разнообразных технологий доступа, многие из которых используются и в приложениях Delphi. Тем не менее, любое приложение баз данных в Delphi имеет стандартное ядро, структура которого определена архитектурой приложения баз данных.

Набор базовых компонентов и способов разработки является единой основой, на которой базируются технологии доступа к данным. Это позволило унифицировать процесс разработки приложений баз данных.

В основе процесса разработки лежит триада компонентов:

- невизуальные компоненты набора данных;
- невизуальные компоненты TDataSource;
- визуальные компоненты отображения данных.

**Форма контроля: контрольные вопросы**

1. Для чего предназначено приложение баз данных?
2. Какими бывают СУБД?
3. Какие механизмы включают приложения базы данных?
4. Что представляет собой источник данных?
5. Поясните особенности VCL Delphi.
6. С помощью каких компонентов создается базовый механизм доступа к данным?
7. Что представляет собой компонент TDataSource?
8. Поясните механизм доступа к данным приложения баз данных.
9. Как создается модуль данных и для чего?
10. Назовите компоненты, используемые для отображения данных.

## Рекомендуемая литература

**Основная литература**

1. Брукс Ф. П. Как проектируются и создаются программные комплексы. М.: Наука, 2009.

2. Липаев В.В. Проектирование программных средств. -М.:Высшая школа,1990.

3. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2007 г.

4. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.

5. Федоров А., Елманова Н. Введение в базы данных. Настольные СУБД – КомпьютерПресс, 2000, №4.

**Дополнительная литература**

6. Дарахвелидзе П.Г., Марков Е.П. Программирование в Delphi7. Спб.: БХВ - Санкт-Петербург,2003.-784с.:ил.

7. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».

## Тема 2
## Набор данных

### План:

1. Абстрактный набор данных
2. Стандартные компоненты
3. Индексы в наборе данных
4. Состояния набора данных

Любое приложение баз данных должно уметь выполнять как минимум две операции. Во-первых, иметь информацию о местонахождении базы данных, подключаться к ней и считывать имеющуюся в таблицах БД информацию. Эта функция в значительной степени зависит от реализации конкретной технологии доступа к данным.

Во-вторых, обеспечивать представление и редактирование полученных данных. Множество записей одной или нескольких таблиц, переданные в приложение в результате активизации компонента доступа к данным, будем называть набором данных. Понятно, что в объектно-ориентированной среде для представления какой-либо группы записей приложение должно использовать возможности некоторого класса. Этот класс должен инкапсулировать набор данных и обладать методами для управления записями и полями.

Таким образом, сам набор данных и класс набора данных является той осью, вокруг которой вращается любая деятельность приложения баз данных.

Пользователь просматривает на экране данные - это результат использования набора данных.

Пользователь решил изменить какое-то число - он изменит содержимое ячейки набора данных.

При закрытии приложение сохраняет все изменения - это набор данных передается в базу данных для сохранения.

При этом, используя одни базовые функции для обслуживания набора данных, компоненты должны обеспечивать доступ к данным в рамках различных технологий. Поэтому не удивительно, что разработчики VCL уделили особое внимание созданию максимально эффективной иерархии классов, обеспечивающих использование наборов данных (рис. 2.1).

Класс TDataset является базовым классом иерархии, он инкапсулирует абстрактный набор данных и реализует максимально общие методы работы с ним. В него можно передать записи из таблицы базы данных или строки из обычного текстового файла — набор данных будет функционировать одинаково хорошо.

На основе базового класса реализованы специальные компоненты VCL для различных технологий доступа к данным, которые позволяют разработчику конструировать приложения баз данных, используя одни и те же приемы и настраивая одинаковые свойства.

В этой лекции рассматриваются следующие вопросы:

- набор данных, инкапсулированный в классе TDataSet;
- что такое состояния набора данных;
- индексы, поля, параметры;
- прототипы компонентов для работы с таблицами, запросами и хранимыми процедурами;
- основные механизмы набора данных, реализованные в классе TDataSet.

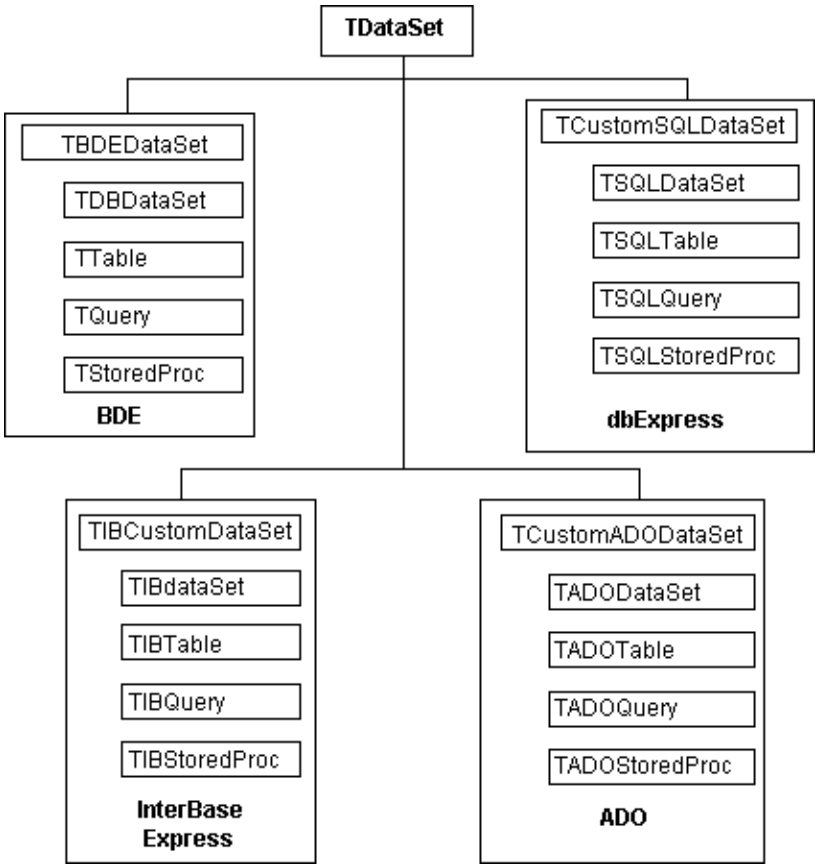


Рисунок 2.1. Иерархия классов, обеспечивающих функционирование набора данных

Абстрактный набор данных

В основе иерархии классов, обеспечивающих функционирование наборов данных в приложениях баз данных Delphi, лежит класс TDataSet. Хотя он почти не содержит методов, реально обеспечивающих работоспособность основных механизмов набора данных, тем не менее, его значение трудно переоценить.

Этот класс задает структурную основу функционирования набора данных. Другими словами, это скелет набора данных, к методам которого необходимо лишь добавить требуемые вызовы соответствующих функций реальных технологий.

При решении наиболее распространенных задач программирования в процессе создания приложений баз данных класс TDataSet не нужен. Тем не менее, знание основных принципов работы набора данных всегда полезно. Кроме этого, класс TDataSet может использоваться разработчиками в качестве основы

для создания собственных компонентов. Поэтому рассмотрим основные механизмы, реализованные в наборе данных.

Набор данных открывается и закрывается свойством

property Active: Boolean;

которому соответственно необходимо присвоить значение True или False. Аналогичные действия выполняют методы

procedure Open;

procedure Close;

После открытия набора данных можно перемещаться по его записям.

На одну запись вперед и назад перемещают курсор соответственно методы

procedure Next;

procedure Prior;

На первую и последнюю запись можно попасть, используя соответственно методы

procedure First;

procedure Last;

Признаком того, что достигнута последняя запись набора, является свойство

property Eof: Boolean;

которое в этом случае имеет значение True.

Аналогичную функцию для первой записи выполняет свойство

property Bof: Boolean;

Перемещение вперед и назад на заданное число записей выполняет метод

function MoveBy(Distance: Integer): Integer;

Параметр Distance определяет число записей. Если параметр отрицательный — перемещение осуществляется к началу набора данных, иначе — к концу.

Для ускоренного перемещения по набору данных можно отключить все связанные компоненты отображения данных. Это делается методом

procedure DisableControls;

Обратная операция выполняется методом

procedure EnableControls;

Общее число записей набора данных возвращает свойство

property RecordCount: Integer;

однако использовать его нужно аккуратно, т. к. каждое обращение к этому свойству приводит к обновлению набора данных, что может вызвать проблемы для больших таблиц или сложных запросов. Если вам нужно определить, не является ли набор данных пустым (часто используемая операция), можно использовать метод

function IsEmpty: Boolean;

который возвращает значение True, если набор данных пуст, или уже упоминавшиеся свойства

```
...
if MyTable.Bof and MyTable.Eof
 then ShowMessage('DataSet is empty');
...
```

Номер текущей записи позволяет узнать свойство

property RecNo: Integer;

Размер записи в байтах возвращает свойство

property RecordSize: Word;

Каждая запись набора данных представляет собой совокупность значений полей таблицы. В зависимости от типа компонента и его настройки, число полей в наборе данных может изменяться. И совсем не обязательно набор данных должен содержать все поля таблицы базы данных.

Совокупность полей набора данных инкапсулирует свойство

property Fields: TFields;

а все необходимые параметры полей содержатся в свойстве

property FieldDefs: TFieldDefs;

Общее число полей набора данных возвращает свойство

property FieldCount: Integer;

а общее число полей типа BLOB содержится в свойстве

property BlobFieldCount: Integer;

Доступ к значениям полей текущей записи предоставляет свойство

property FieldValues[const FieldName: string]: Variant; default;

где в параметре FieldName задается имя поля.

В процессе программирования разработчик очень часто обращается к полям набора данных. Если структура полей набора данных жестко задана и не изменяется, это можно сделать так:

for i := 0 to MyTable.FieldCount - 1 do

MyTable.Fields[i].DiplayFormat := '#.###';

Иначе, если порядок следования полей и их состав меняется, можно использовать метод

function FieldByName(const FieldName: string): TField;

И делается это следующим образом:

MyTable.FieldByName('VENDORNO').Aslnteger := 1234;

Имя поля, передаваемое в параметре FieldName, не чувствительно к регистру символов.

Метод

procedure GetFieldNames(List: TStrings);

вернет в параметр List полный список имен полей набора данных.

Класс TDataSet содержит ряд свойств и методов, которые обеспечивают редактирование набора данных.

Но сначала бывает полезно поинтересоваться, можно ли редактировать набор данных вообще. Это можно сделать при помощи свойства

property CanModify: Boolean;

которое принимает значение True для редактируемых наборов. Перед началом редактирования набор данных нужно перевести в режим редактирования, использовав метод

procedure Edit;

Для сохранения сделанных изменений применяется метод

procedure Post; virtual;

Разработчик может вызывать его самостоятельно, или же метод Post вызывается самим набором данных при переходе на другую запись.

При необходимости все сделанные после последнего вызова метода Post изменения можно отменить методом

procedure Cancel; virtual;

Новая пустая запись добавляется в конец набора данных методом

procedure Append;

Новая пустая запись добавляется на место текущей методом

procedure Insert;

а текущая запись и все нижеследующие смещаются на одну позицию вниз.

**Внимание**

*При использовании методов Append и insert набор данных переходит в режим редактирования самостоятельно.*

Дополнительно, у вас есть возможность добавить или вставить новую запись уже с заполненными полями. Для этого применяются методы

procedure AppendRecord(const Values: array of const);

procedure InsertRecord(const Values: array of const);

А делается это примерно так:

MyTable.AppendRecord([2345, 'New customer', '+7(812)4569012', 0, '']);

После вызова этих методов и их завершения набор данных автоматически возвращается в состояние просмотра.

Для существующей записи аналогичным образом можно заполнить все поля, использовав метод

procedure SetFields(const Values: array of const);

Текущая запись удаляется методом

procedure Delete;

При этом набор данных не выдает никаких предупреждений, а просто делает это.

Очистить содержимое всех полей текущей записи может метод

procedure ClearFields;

Обратите внимание, что поля становятся пустыми (NULL), а не сбрасываются в нулевое значение.

О том, редактировалась ли текущая запись, сообщает свойство

property Modified: Boolean;

если оно имеет значение True.

Набор данных можно обновить, не закрывая и не открывая его снова. Для этого применяется метод

procedure Refresh;

Однако он сработает только для таблиц и тех запросов, которые нельзя редактировать.

В каждый момент времени набор данных находится в определенном состоянии (о состояниях см. ниже в этой главе). Свойство

type TDataSetState = (dslnactive, dsBrowse, dsEdit, dslnsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue, dsBlockRead, dsInternalCalc, dsOpening); property State: TDataSetState;

дает информацию о текущем состоянии набора.

Методы-обработчики класса TDataSet предоставляют разработчику широчайшие возможности по отслеживанию событий, происходящих с набором данных.

По паре методов-обработчиков (до и после события) предусмотрено для следующих событий в наборе данных:

- открытие и закрытие набора данных;
- переход в режим редактирования;
- переход в режим вставки новой записи;
- сохранение сделанных изменений;
- отмена сделанных изменений;
- перемещение по записям набора данных;
- обновление набора данных.

Обратите внимание, что помимо методов-обработчиков режима вставки существует дополнительный метод

property OnNewRecord: TDataSetNotifyEvent;

который вызывается непосредственно при вставке или добавлении записи. Дополнительно к этому могут использоваться методы-обработчики возникающих ошибок. Они предусмотрены для ошибок удаления, редактирования и сохранения изменений.

Метод-обработчик

property OnCalcFields: TDataSetNotifyEvent;

очень важен для задания значений вычисляемых полей. Он вызывается для каждой записи, которая отображается в визуальных компонентах, связанных с набором данных каждый раз, когда необходимо перерисовать значения полей в визуальных компонентах.

Если в методе-обработчике OnCalcFields производятся слишком сложные вычисления, частота его вызовов может быть уменьшена за счет свойства

property AutoCalcFields: Boolean;

По умолчанию оно равно значению True и расчет вычисляемых полей производится при каждой перерисовке. При значении False метод-обработчик OnCalcFields вызывается только при открытии, переходе в состояние редактирования и обновлении набора данных. Все перечисленные выше обработчики имеют одинаковый тип

type TDataSetNotifyEvent = procedure(DataSet: TDataSet) of object;

И метод-обработчик

type TFilterRecordEvent = procedure(DataSet: TDataSet;

var Accept: Boolean) of object;

property OnFilterRecord: TFilterRecordEvent;

вызывается для каждой записи набора данных при свойстве Filtered = True. Помимо перечисленных, класс TDataSet содержит еще много свойств и методов, которые обеспечивают работоспособность многих полезных в практическом программировании приложений баз данных функций.

Стандартные компоненты

Можно заметить, что на рис. 2.1 набор компонентов для каждой из представленных технологий доступа к данным примерно одинаков. Везде есть компонент, инкапсулирующий табличные функции, компонент запроса SQL и компонент хранимой процедуры. И хотя все они имеют разных ближайших предков, тем не менее, функциональность подобных компонентов в различных технологиях почти одинакова.

Поэтому имеет смысл рассмотреть общие для компонентов свойства и методы, представив, что существуют некие виртуальные общие предки для таблицы, запроса и хранимой процедуры.

**Примечание**

*Некоторые из описываемых ниже свойств и методов присутствуют не в каждой реализации компонентов.*

**Компонент таблицы**

Компонент таблицы обеспечивает доступ к таблице базы данных целиком, создавая набор данных, структура полей которого полностью повторяет таблицу БД. За счет этого компонент прост в настройке и обладает многими дополнительными функциями, которые обеспечивают применение табличных индексов.

Но в практике программирования работа с таблицами целиком используется не так часто. А при работе с серверами баз данных промежуточное ПО, используемых технологий доступа к данным, все равно транслирует запрос на получение табличного набора данных в простейший запрос SQL, например:

SELECT * FROM Orders

В такой ситуации применение табличных компонентов становится менее эффективным, чем использование запросов.

После соединения с источником данных необходимо задать имя таблицы в свойстве

property TableName: String;

Иногда в свойстве TableType дополнительно задается тип таблицы.

Если соединение с источником данных настроено правильно, имя таблицы можно выбрать из выпадающего списка свойства TableName.

Преимуществом табличного компонента является использование индексов, которые ускоряют работу с таблицей. Все индексы, созданные в базе данных для таблицы, автоматически загружаются в компонент. Их параметры доступны через свойство

property IndexDefs: TIndexDefs;

Подробно класс TIndexDefs рассматривается ниже в этой главе.

При работе с компонентом разработчик имеет возможность управлять индексами.

Существующий индекс можно выбрать в Инспекторе объектов в списке свойств

property IndexName: String;

или использовать свойство

property IndexFieldNames: String;

в котором можно задать произвольное сочетание имен индексированных полей таблицы. Имена полей разделяются символом точкой с запятой. Таким образом, при помощи свойства IndexFieldNames можно создавать составные индексы.

Свойства IndexName и IndexFieldNames нельзя использовать одновременно.

Число полей, используемых в текущем индексе табличного компонента, возвращает свойство

property IndexFieldCcunt: Integer;

А свойство

property IndexFields: [Index: Integer]: TField;

представляет собой индексированный список полей, входящих в текущий индекс:

for i := 0 to MyTable.IndexFieldCount — 1 do MyTable.IndexFields[i].Enabled := False;

Для выполнения операций с таблицами и индексами целиком в табличных компонентах реализовано несколько методов.

Метод

procedure CreateTable;

создает новую таблицу в базе данных, используя заданное имя и описание полей, и индексов из свойств TFieldDefs и TindexDefs. Если таблица с таким именем уже имеется в базе данных, то она будет уничтожена и создана заново с новой структурой и данными.

Метод

procedure EmptyTable;

удаляет из набора данных и таблицы базы данных все записи.

Метод

procedure DeleteTable;

уничтожает таблицу базы данных, связанную с компонентом. Набор данных должен быть закрыт.

Метод

type

TIndexOption = (ixPrimary, ixUnique, ixDescending, ixCaselnsensitive,

ixExpression, ixNonMaintained);

TIndexOptions = set of TIndexOption;

procedure Addlndex(const Name, Fields: String; Options: TIndexOptions, const DescFields: String='');

добавляет к таблице БД новый индекс. Параметр Name задает имя индекса. В параметре Fields через точку с запятой определяются имена полей, входящих в индекс. Параметр DescFields задает описание индекса из констант, объявленных в типе TIndexOption.

Метод

procedure Deletelndex(const Name: string);

уничтожает индекс.

**Компонент запроса**

Компонент запроса предназначен для создания запроса SQL, подготовки его параметров, передачи запроса на сервер БД и представления результата запроса в наборе данных. При этом набор данных может быть редактируемым или нет.

Любой компонент запроса, каждая строка набора данных которого однозначно связывается с одной строкой таблицы БД, может редактироваться. Например, запрос

SELECT * FROM Country

редактировать можно. Если же приведенное правило не выполняется, то набор данных можно использовать только для просмотра, и, конечно, возможности компонентов здесь ни при чем. Куда, к примеру, записывать результаты редактирования записей следующего запроса:

```
SELECT CustNo, SUM(AmountPaid)
FROM Orders
 GROUP BY CustNo
```

Ведь в таком запросе каждая запись есть результат суммирования неизвестного заранее числа других записей.

Тем не менее, компоненты запросов предоставляют разработчику мощный и гибкий механизм работы с данными. С помощью компонентов запросов можно решать гораздо более сложные задачи, чем с табличными компонентами.

В целом компонент запроса работает быстрее, т. к. структура возвращаемых запросом полей может изменяться, то экземпляры класса TFieldDef, хранящие информацию о свойствах полей, создаются по необходимости при запуске приложения. Табличный компонент создает все классы для описания полей в любом случае, поэтому в приложениях клиент-сервер табличный компонент может открываться медленнее, чем запрос. Рассмотрим общие свойства и методы компонентов запросов. Текст запроса определяется свойством

```
property SQL: TStrings;
```

В свойстве

```
property Text: PChar;
```

содержится окончательно подготовленный текст запроса перед пересылкой его на сервер.

Выполнение запроса возможно тремя способами.

Если запрос возвращает результат в набор данных, то применяется метод

```
procedure Open;
```

или свойство

```
property Active: Boolean;
```

которому присваивается значение True. После выполнения запроса открывается набор данных компонента. Закрывается такой запрос методом

```
procedure Close;
```

или тем же свойством Active.

Если запрос не возвращает результат в набор данных (например, использует операторы INSERT, DELETE, UPDATE), то используется метод

```
procedure ExecSQL;
```

и после выполнения запроса набор данных компонента не открывается.

Попытка использовать для такого запроса метод open или свойство Active приведет к ошибке создания указателя на курсор данных.

После выполнения запроса в свойстве

property RowsAffected: Integer;

возвращается число обработанных при выполнении запроса записей.

Для того чтобы разрешить редактирование набора данных запроса, необходимо свойству

property RequestLive: Boolean;

присвоить значение True. Это свойство устанавливается, но не работает для запроса, результат которого не модифицируется из-за самого запроса.

Для подготовки запроса к выполнению предназначен метод

procedure Prepare;

который обеспечивает выделение необходимых ресурсов на сервере и проведение оптимизации.

Метод

procedure UnPrepare;

освобождает занятые при подготовке запроса ресурсы.

Результат выполнения этих двух операций отражается в свойстве

property Prepared: Boolean;

Значение True данного свойства говорит о том, что запрос подготовлен для выполнения.

Вызов методов Prepare и UPPrepare не является обязательным, т. к. компонент делает это автоматически. Однако если запрос будет выполняться несколько раз подряд, то подготовку необходимо провести перед первым выполнением запроса вручную. Тогда при последующих выполнениях сервер не будет тратить время на проведение бесполезной операции — ведь ресурсы под запрос уже были выделены.

Часто запросы имеют настраиваемые параметры, значения которых определяются непосредственно перед выполнением запроса.

Свойство

property Params: TParams;

представляет собой список объектов TParams, каждый из которых содержит настройки одного параметра. Свойство Params обновляется автоматически при изменении текста запроса. Подробнее о классе TParams рассказывается ниже в этой главе.

**Примечание**

*В компоненте TADOQuery свойство, аналогичное описанному свойству Params, называется Parameters.*

Свойство

property ParamCount: Word;

возвращает число параметров запроса.

Свойство

property ParamCheck: Boolean;

определяет, необходимо ли обновлять свойство Params при изменении текста запроса во время выполнения. При значении True обновление осуществляется.

**Компонент хранимой процедуры**

Компонент хранимой процедуры предназначен для определения процедуры, установки ее параметров, выполнения процедуры и возвращения результатов в компонент.

В зависимости от выбранной технологии доступа к данным, каждый компонент хранимой процедуры имеет собственный способ соединения с сервером. После подключения к источнику данных имя хранимой процедуры можно выбрать из списка свойства

property StoredProcName: String;

После этого свойство

property Params: TParams;

предназначенное для хранения параметров процедуры, автоматически заполняется.

Для хранимых процедур важно деление параметров на входные и выходные. Первые содержат исходные данные, а вторые передают результаты выполнения процедуры.

Детально класс TParams описывается ниже. Общее число параметров возвращает свойство

property ParamCount: Word;

Для подготовки хранимой процедуры используется метод

procedure Prepare;

или свойство

property Prepared: Boolean;

которое должно получить значение True.

Метод

procedure UnPrepare;

или свойство

Prepared := False

выполняют обратное действие.

Кроме того, проверка значения свойства Prepared позволяет установить, осуществлялась ли подготовка процедуры к выполнению или нет.

**Внимание**

*После выполнения хранимой процедуры исходный порядок следования параметров в списке Params может измениться. Поэтому для доступа к конкретному параметру рекомендуется использовать метод*

function ParamByName(const Value: String): TParam;

Если хранимая процедура возвращает набор данных, компонент можно открывать методом

procedure Open;

или свойством

property Active: Boolean;

В противном случае для выполнения процедуры используется метод

procedure ExecProc;

и после этого выходные параметры получат вычисленные значения.

Индексы в наборе данных

Важнейшей проблемой для любой БД является достижение максимальной производительности и ее сохранение при дальнейшем увеличении объемов хранимых данных. Использование индексов позволяет решить эту задачу.

Индекс представляет собой часть базы данных, в которой содержится информация об организации данных в таблицах БД.

В отличие от ключей, которые просто идентифицируют отдельные записи, индексы занимают дополнительные объемы памяти (довольно значительные) и могут храниться как совместно с таблицами, так и в виде отдельных файлов. Индексы создаются вместе со своей таблицей и обновляются при модификации данных. При этом работа по обновлению индекса для большой таблицы может отнимать много ресурсов, поэтому имеет смысл ограничить число индексов для таких таблиц, где происходит частое обновление данных.

Индекс содержит в себе уникальные идентификаторы записей и дополнительную информацию об организации данных. Поэтому если при выполнении запроса сервер или локальная СУБД обращается для отбора записи к индексу, то это занимает значительно меньше времени, т. к. понятно, что идентификатор гораздо меньше самой записи. Кроме этого, индекс "знает", как организованы данные и может ускорять обработку за счет группирования записей по сходным значениям параметров.

Создание для БД эффективного набора индексов является нетривиальной задачей.

Во-первых, нужно верно определить оптимальное число индексов для каждой таблицы. Во-вторых, каждый индекс должен содержать только необходимые поля, при этом большую роль играет их упорядочивание.

В большинстве СУБД при создании индексов требуется только задать поля и название индекса, вся остальная работа выполняется автоматически.

Естественно, что в компонентах доступа к данным VCL Delphi используются все возможности такого мощного инструмента, как индексы. Причем свойства и методы для работы с индексами присутствуют только в табличных компонентах, т. к. в компонентах запросов работа с индексами осуществляется средствами SQL.

Набор данных может работать и без применения индексов, но для этого соответствующая таблица БД не должна иметь первичного ключа — случай довольно редкий. Поэтому по умолчанию в наборе данных используется первичный индекс. При открытии набора данных все записи отсортированы в соответствии с первичным ключом.

**Механизм подключения индексов**

Для того чтобы подключить к набору данных вторичный индекс, необходимо присвоить свойству indexName название индекса. Если свойство не имеет значения, то в наборе данных используется первичный индекс.

Альтернативный способ задания индекса заключается в использовании свойства indexFieldNames, в котором задается перечень имен полей необходимого индекса, разделенных точкой с запятой. При этом в Инспекторе объектов для данного свойства список полей существующих индексов создается автоматически, разработчику остается сделать выбор. При помощи свойства

indexFieldNames можно создавать и составные индексы. Для этого необходимо, чтобы все входящие в список поля были индексированы.

Список имен всех индексов можно получить при помощи метода

GetlndexNames.

**Примечание**

*Изменение текущего индекса можно осуществлять без отключения набора данных, поэтому в приложениях очень удобно делать сортировку данных по индексам. Такой метод смены индексов называется индексацией "на лету".*

После установки индекса количество полей в индексе передается в свойство

IndexFieldCount.

**Список описаний индексов**

Информация об индексах набора данных содержится в свойстве класса

TDataSet

property IndexDefs: TindexDefs;

В нем для каждого индекса создается структура TindexDef. Доступ к информации об индексах осуществляется через свойство

property Items[Index: Integer]: TindexDef; default;

являющееся списком объектов TindexDef.

Объекты типа TindexDef можно добавлять в список при помощи метода

function AddlndexDef: TindexDef;

Поиск объекта описания индекса осуществляет метод

function Find(const Name: String): TindexDef;

который возвращает найденный объект по заданному в параметре Name имени индекса.

Пара методов

function FindlndexForFields(const Fields: string): TindexDef;

function GetlndexForFields(const Fields: String;

Caselnsensitive: Boolean): TindexDef;

находит объект описания индекса по списку полей, входящих в индекс. Если индекс не найден, ищется первый индекс, начинающийся с указанных полей. Первый из этих двух методов в случае неудачного поиска генерирует исключительную ситуацию EDatabaseError, а второй возвращает nil.

Список IndexDefs обновляется автоматически при открытии набора данных. Но метод

procedure Update; reintroduce;

обновляет список описаний индексов без открытия набора данных.

**Описание индекса**

Параметры каждого индекса набора данных представлены в классе TindexDef, а их совокупность для набора данных содержится в свойстве IndexDefs класса TDataSet.

Свойство

property Name: String;

определяет название индекса.

Список всех полей индекса содержится в свойстве

property Fields: String;

Поля разделяются точкой с запятой.

Свойство

property CaselnsFields: String;

содержит список полей, регистр символов в которых при сортировке не учитывается. Поля разделяются точкой с запятой. Все поля из этого списка должны входить в свойство Fields. В наборе данных по умолчанию используется сортировка записей с учетом регистра символов. Но некоторые серверы БД допускают комбинированную сортировку по полям с учетом регистра и без.

Свойство

property DescFields: String;

содержит список полей через точку с запятой, которые сортируются в обратном порядке. Все поля из этого списка должны входить в свойство Fields. По умолчанию все поля сортируются в прямом порядке. Некоторые серверы БД поддерживают одновременную сортировку полей в прямом и обратном порядке.

Свойство

property GroupingLevel: Integer;

позволяет ограничить область применения индекса. Если значение этого свойства равно нулю, индекс упорядочивает все записи набора данных. В противном случае действие индекса распространяется на группы записей, имеющих одинаковые значения для того числа полей, которое задано этим свойством.

Параметры индекса определяются свойством

property Options: TIndexOptions;

Для индекса возможны сочетания следующих параметров:

- ixPrimary — первичный индекс;
- ixunique — значения индекса уникальны;
- ixDescending — индекс сортирует записи в обратном порядке;
- ixCaseinsensitive — индекс сортирует записи без учета регистра символов;
- ixExpression — в индексе используется выражение (для индексов dBASE);
- ixNonMaintained — индекс не обновляется при открытии таблицы.

Метод

procedure Assign(ASource: TPersistent); override;

заполняет свойства объекта значениями аналогичных свойств объекта ASource.

**Использование описаний индексов**

Описания индексов наряду с описаниями полей также используются при создании новых таблиц БД. Для каждого планируемого индекса перед вызовом метода CreateTable необходимо создать или скопировать из существующего набора данных соответствующее описание. Тогда при создании таблицы индексы будут добавлены автоматически:

```
with Tablel do
 begin
DatabaseName := 'DBDEMOS';
TableType := ttParadox;
TableName := 'DemoTable';
```

```
...
{Создание описаний полей}
...
with IndexDefs do begin Clear;
AddlndexDef; with Items[0] do
begin
Name := ' ' ; Fields := 'Fieldl'; Options := [ixPrimary, ixUnique];
  end;
AddlndexDef; with Items[1] do
 begin
Name := 'Secondlndex'; Fields := 'Fieldl;Field2';
Options := [ixCaselnsensitive];
end;
 end;
CreateTable;
end;
```

При создании описаний индексов использован метод AddlndexDef, который при каждом вызове добавляет к списку Items объекта TIndexDefs новый объект TindexDef. Таким образом сначала создается первичный индекс (в таблицах Paradox он не имеет имени), затем вторичный индекс SecondI"ndex. Для каждого описания обязательно определяются составляющие индекс поля и параметры индекса (свойства Fields и options).

**Параметры запросов и хранимых процедур**

Свойство Params представляет собой набор изменяемых параметров запроса или хранимой процедуры, а также набор объектов TParam, инкапсулирующих отдельные параметры.

Рассмотрим следующий запрос SQL:

```
SELECT SaleDat, OrderNo
FROM Orders
WHERE SaleDat >= '01.08.2001' AND SaleDat <= '31.08.2001'
```

В нем осуществляется отбор номеров заказов, сделанных в августе 2001 года. Теперь вполне естественно было бы предположить, что пользователю может понадобиться получить подобный отчет за другой месяц или за первые десять дней августа.

В этом случае можно поступить так:

```
procedure TForml.FormCreate(Sender: TObject);
begin
with Queryl do
 begin
SQL[0] := 'SELECT PartDat, ItemNo, ItemCount, InputPrice';
SQL[1] := 'FROM Parts';
SQL[2] := 'WHERE PartDat>= "01.08.2001" AND PartDat<=" 31. 08 . 2001 '";
 end;
 end;
procedure TForml.RunBtnClick(Sender: TObject);
```

```
 begin
with Queryl do
begin
if Active then Close;
SQL[2] := 'WHERE PartDat>= '+chr(39)+DatelEdit.Text+chr(39)+
AND PartDat<='+chr(39)+Date2Edit.Text+chr(39);
 Open;
  end;
end;
```

При создании формы в методе FormCreate задается текст запроса. Для этого используется свойство SQL. При щелчке на кнопке RunBtn, в соответствии с заданными в однострочных редакторах DatelEdit и Date2Edit датах, изменяется текст запроса. Метод FormCreate приведен только для того, чтобы обозначить первоначальный текст запроса, этот текст вполне можно задать в свойстве SQL.

Для решения подобных задач как раз и используются параметры. В этом случае текст запроса будет выглядеть следующим образом:

```
SELECT PartDat, ItemNo, ItemCount, InputPrice
FROM Parts
WHERE PartDat>= :PD1 AND PartDat<= :PD2
```

Двоеточие перед именами PD1 и PD2 означает, что это параметры. Имя параметра выбирается произвольно. В списке свойства Params первым идет тот параметр, который расположен первым по тексту запросу.

После ввода в свойстве SQL текста запроса для каждого параметра автоматически создается объект TParam. Эти объекты доступны в специализированном редакторе, который вызывается при щелчке на кнопке свойства Params в Инспекторе объектов (рис. 2.2). Для каждого параметра требуется установить тип данных, который должен согласовываться с типом данных соответствующего поля.

Теперь для задания текущих ограничений по дате поступления можно использовать свойство Params:

```
procedure TForml.RunBtnClick(Sender; TObject);
begin
with Queryl do
begin
  Close;
Params[0].AsDateTime := StrToDate(DatelEdit.Text);
  Params[l].AsDateTime := StrToDate(Date2Edit.Text);
Open;
end;
end;
```

При щелчке на кнопке RunBtn при помощи параметров в запрос передаются текущие значения ограничений дат.

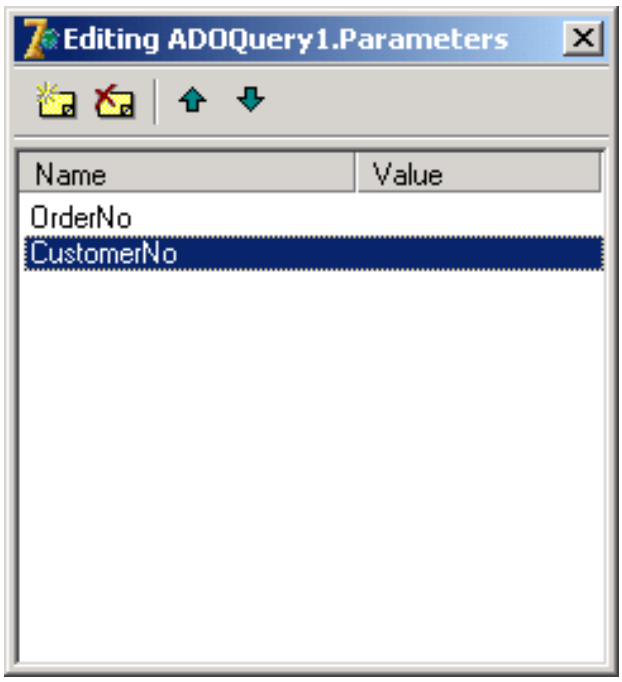


Рисунок 2.2. Специализированный редактор параметров запроса

Значения параметров запроса можно задать и из другого набора данных. Для этого применяется свойство DataSource компонента набора данных. Указанный в свойстве компонент TDataSource должен быть связан с набором данных, значения полей которого требуется передать в параметры. Названия параметров должны соответствовать названиям полей этого набора данных, тогда свойство DataSource начнет работать. При перемещении по записям набора данных текущие значения одноименных параметров полей автоматически передаются в запрос.

Для иллюстрации работы этого свойства рассмотрим простой пример, главная форма которого представлена на рис. 2.3. Этот проект не имеет ни одной строки написанного вручную программного кода.

Верхний компонент TDBGrid отображает данные из таблицы Orders базы данных DBDEMOS и связан через компонент OrdersSource типа TDataSource с компонентом запроса ordersQuery. Текст запроса выглядит так:

SELECT * FROM Orders

Нижний компонент TDBGrid отображает данные о покупателе и через компонент CustSource типа TDataSource связан с запросом CustSource, свойство SQL которого имеет следующий вид:

SELECT * FROM Customer WHERE CustNo=:CustNo

Обратите внимание, что название параметра соответствует названию поля номера покупателя из таблицы Orders.

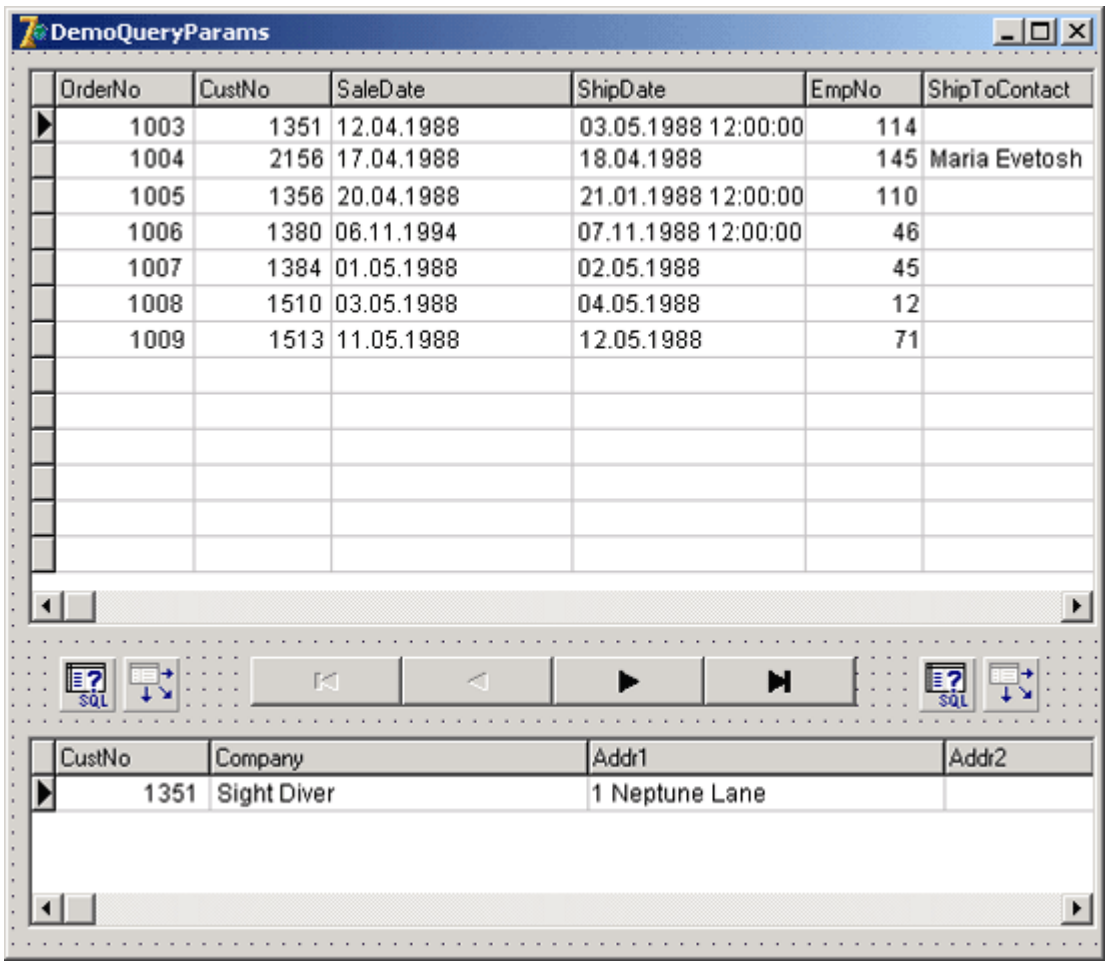


Рисунок 2.3. Главная форма проекта DemoQueryParams

Свойство DataSource компонента custQuery указывает на компонент OrdersSource.

Как только оба набора данных открываются, текущее значение из поля CustNo набора данных orders автоматически передается в параметр запроса компонента CustQuery.

Благодаря этому, для двух наборов данных реализована связь "один-к-одному" по полю номера поставщика.

И в завершение разговора о параметрах запросов рассмотрим свойства и методы класса TParams, составляющего свойство Params, и класса TParam, инкапсулирующего отдельный параметр.

**Класс TParams**

Класс TParams представляет собой список параметров.

Доступ к элементам списка возможен через индексированное свойство

property Items[Index: Word]: TParam;

а к значениям параметров — через свойство

property ParamValues[const ParamName: String]: Variant;

Добавить новый параметр можно методом

procedure AddParam(Value: TParam);

Но для него необходимо создать объект параметра. Это можно сделать методом

function CreateParamfFldType: TFieldType; c

onst ParamName: string;

ParamType: TParamType): TParam;

где FidType — тип данных параметра, ParamName — имя параметра и ParamType — тип параметра (см. ниже).

И оба метода можно использовать в связке:

MyParams.AddParam(MyParams.CreateParam(ftInteger, 'Paraml', ptInput));

Вместо того, чтобы заполнять параметры по одному, можно использовать метод

function ParseSQL(SQL: String; DoCreate: Boolean): String;

который при DoCreate = True анализирует текст запроса из свойства SQL и создает новый список параметров.

Или же, для присвоения значений сразу всем параметрам используется метод

procedure AssignValues(Value: TParams);

Для удаления параметра из списка применяется метод

procedure RemoveParam(Value: TParam);

При работе с параметрами для их идентификации полезно использовать обращение по имени, т. к. при работе с хранимыми процедурами после их выполнения порядок следования может измениться. Также и при использовании динамических запросов (их текст SQL может изменяться во время выполнения).

Для обращения к параметру по имени используется метод

function ParamByName(const Value: String): TParam;

В сложных запросах SQL или после многочисленных исправлений разработчик может допустить ошибку и создать два разных параметра с одним именем. В этом случае при выполнении запроса одноименные параметры считаются одним и им присваиваются значение первого по порядку запроса. Для контроля повторных имен в списке параметра используется метод

function IsEqual(Value: TParams): Boolean;

который возвращает значение True, если для параметра value найден дубликат.

**Класс TParam**

Класс TParam инкапсулирует свойства отдельного параметра. Имя параметра определяется свойством

property Name: String;

Тип данных параметра задает свойство

property DataType: TFieldType;

Тип данных параметра и связанного поля должны совпадать.

Тип параметра определяется множеством

type

TParamType = (ptUnknown, ptInput, ptOutput, ptlnputOutput, ptResult);

TParamTypes = set of TParamType;

которое имеет следующие значения:

- ptUnknown — тип неизвестен;
- ptinput — параметр предназначен для передачи значения из приложения;
- ptOutput — параметр предназначен для передачи значения в приложение;

- ptlnputOutput — параметр предназначен для передачи и приема значения;
- ptResult — параметр предназначен для передачи в приложения информации о статусе операции.

Свойство

property ParamType: TParamType;

определяет тип параметра.

При работе с параметрами довольно часто бывает необходимо определить, имеет ли параметр ненулевое значение. Для этого используется свойство

property IsNull: Boolean;

Свойство возвращает значение True, если параметр не имеет значения или имеет значение Null.

Свойство

property Bound: Boolean;

возвращает значение True только тогда, когда параметру не присваивалось значение вообще.

Метод

procedure Clear;

присваивает параметру значение Null.

Само значение параметра задается свойством

property Value: Variant;

Но использование вариантов не очень эффективно, когда требуется обеспечить максимальную скорость. В таких случаях можно обратиться к целому набору свойств AS ..., которые не только возвращают значение, но и приводят его к некоторому типу. Например, свойство

property Aslnteger: Longlnt;

возвращает целочисленное значение поля.

**Примечание**

*Необходимо осторожно использовать свойства с приведением типа, т. к. попытка преобразования неверного значения вызовет исключительную ситуацию.*

Для чтения из буфера и записи в буфер значения параметра соответственно используются методы

procedure SetData(Buffer: Pointer);

procedure GetData(Buffer: Pointer);

а необходимый размер при записи в буфер позволит определить метод

function GetDataSize: Integer;

Можно скопировать тип данных, имя и значение параметра прямо из поля данных. Для этого применяется метод

procedure AssignField(Field: TField);

а для присвоения типа данных и значения используется метод

procedure AssignFieldValue(Field: TField; const Value: Variant);

Общее число знаков для числовых значений определяет свойство

property Precision: Integer;

А свойство

property NumericScale: Integer;

задает число знаков после запятой.

Для строковых параметров размер задает свойство

property Size: Integer;

Состояния набора данных

В процессе своего функционирования (от открытия методом Open и до закрытия методом close) набор данных может выполнять самые разнообразные операции. Можно просто перемещаться по записям, можно редактировать данные и удалять записи, можно проводить поиск по различным параметрам и т. д. При этом желательно, чтобы все операции выполнялись как можно быстрее и эффективнее.

Набор данных в любой момент времени находится в некотором состоянии, т. е. подготовлен к выполнению действий строго определенного рода. И для каждой группы операций набор данных выполняет ряд подготовительных действий.

Все состояния набора данных делятся на две группы.

- К первой группе относятся состояния, в которые набор данных переходит автоматически, а также непродолжительные по времени состояния, сопровождающие функционирование полей набора данных (табл. 2.1).
- Во вторую группу входят состояния, которыми можно управлять из приложения, например, перевод набора данных в режим редактирования (табл. 2.2).

Базовый класс TDataSet, инкапсулирующий свойства набора данных, позволяет изменять состояние, а также проверять текущее состояние набора данных.

Текущее состояние набора данных передается в свойство state, имеющее тип TDataSetState:

type TDataSetState = (dslnactive, dsBrowse, dsEdit, dslnsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue, dsBlockRead, dsInternalCalc);

Для управления состояниями набора данных используются методы open, Close, Edit, Insert.

Таблица 12.1. Автоматические состояния набора данных

| **Константа состояния** | **Описание** |
|---|---|
| dsNewValue | Включается при обращении к свойству NewValue поля набора данных |
| dsOldValue | Включается при обращении к свойству OldValue поля набора данных |
| dsCurValue | Включается при обращении к свойству CurValue поля набора данных |
| dsInternalCalc | Включается при расчете значений полей, для которых FindKind = fklnternalCalc |

| | |
|---|---|
| dsCalcFields | Включается при выполнении метода onCalcFields |
| dsBlockRead | Включается механизм ускоренного перемещения по набору данных |
| dsOpening | Существует при открытии набора данных методом Open или свойством Active |
| dsFilter | Включается при выполнении метода OnFilterRecord |

Таблица 2.2. Управляемые состояния набора данных

| **Константа состояния** | **Метод** | **Описание** |
|---|---|---|
| dslnactive | Close | Набор данных закрыт |
| dsBrowse | Open | Данные доступны для просмотра, но недоступны для редактирования |
| dsEdit | Edit | Данные можно редактировать |
| dslnsert | Insert | К набору данных можно добавлять новые записи |
| dsSetKey | SetKey | Включается механизм поиска по ключу. Также могут использоваться диапазоны |

Рассмотрим, как изменяется состояние набора данных при выполнении стандартных операций.

Закрытый набор данных всегда имеет неактивное состояние dsinactive.

При открытии набор данных переходит в состояние просмотра данных dsBrowse. В этом состоянии по записям набора данных можно перемещаться и просматривать их содержимое, но редактировать данные нельзя. Это основное состояние открытого набора данных, из него можно перейти в другие состояния, но любое изменение состояния происходит через просмотр данных (рис. 2.4).

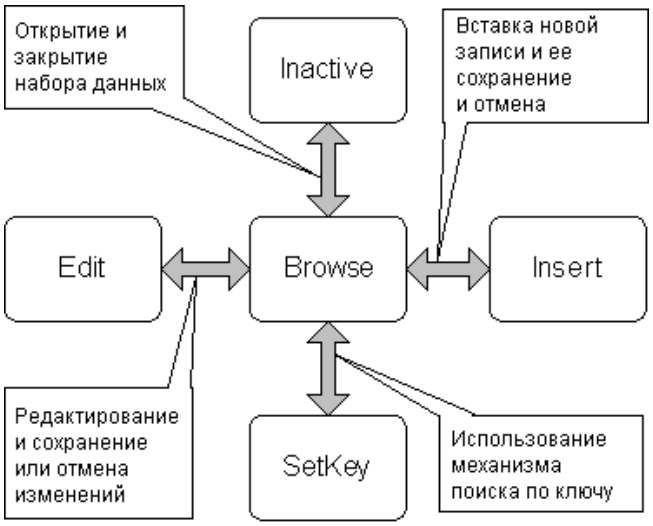

Рисунок 2.4. Схема изменения состояний набора данных

При необходимости редактирования данных набор должен быть переведен в состояние редактирования dsEdit, для этого используется метод Edit. После выполнения метода можно изменять значения полей для текущей записи. При перемещении на следующую запись набор данных автоматически переходит в состояние просмотра.

Для того чтобы вставить в набор данных новую запись, необходимо использовать состояние вставки dsinsert. Метод insert переводит набор данных в это состояние и добавляет на месте текущего курсора новую пустую запись. При переходе на другую запись, после проверки на уникальность первичного ключа (если он есть) набор данных возвращается в состояние просмотра.

Состояние установки ключа dsSetKey используется только в табличных компонентах при необходимости поиска методами FindKey и FindNext, а также при использовании диапазонов (метод setRange). Это состояние сохраняется до момента вызова одного из методов поиска по ключу или метода отмены диапазона. После этого набор данных возвращается в состояние просмотра.

Состояние просмотра по блокам dsBlockRead используется набором данных при реализации быстрого перемещения по большим массивам записей без показа промежуточных записей в компонентах отображения данных и без вызова обработчика события перемещения по записям. Для реализации быстрого перемещения по набору данных можно использовать методы DisableControls И EnableControls.

Резюме

Набор данных является образом таблицы базы данных в приложении. Он содержит группу записей и обеспечивает их использование.

Класс TDataSet, инкапсулирующий функциональность набора данных, является базовым классом для всех технологий доступа к данным. На его основе созданы все основные компоненты, применяемые при разработке приложений баз данных. Условно их можно разделить на три группы:

- компоненты таблиц;
- компоненты запросов;
- компоненты хранимых процедур.

В этой главе рассмотрены важнейшие свойства, методы и структуры, реализованные в компонентах, инкапсулирующих набор данных.

**Форма контроля: контрольные вопросы**

1. Что является основой деятельности приложения баз данных?
2. Особенности класса Класс TDataset.
3. Назовите набор компонентов для каждой из технологий доступа к набору данных.
4. Назначение компонента TTable.
5. Методы, используемые для работы с компонентом TTable.
6. Поясните назначение запроса SQL.
7. Какие компоненты необходимы для создания запросов?

8. Для чего нужны индексы в базе данных?
Поясните механизм подключения индексов.
9. Назовите параметры запросов и хранимых процедур.
10. Какие бывают виды состояний набора данных?

**Рекомендуемая литература**

1. Брукс Ф. П. Как проектируются и создаются программные комплексы. М.: Наука, 2009.
2. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.
3. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2007 г.
4. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.
5. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.
6. Федоров А., Елманова Н. Введение в базы данных. Настольные СУБД – КомпьютерПресс, 2000, №4.

**Дополнительная литература**

7. С.Бобровский. Delphi 7. Учебный курс. Издательство «ПИТЕР»,2005г.
8. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».

## Тема № 3
## Поля и типы данных

**План:**
1. Объекты полей
2. Статические и динамические поля
3. Виды полей
4. Типы данных
5. Ограничения

Каждая таблица БД и, следовательно, каждый набор данных приложения имеет собственную структуру, которая определяется совокупностью полей. Каждое поле набора данных представляет собой объект, содержащий описание типа данных, которому должно соответствовать значение, находящееся в записи на определенном месте. Иначе, полем можно назвать совокупность ячеек с

данными конкретного типа, расположенных в одном и том же месте каждой записи набора данных, или попросту - это столбец в таблице.

В наборе данных приложения баз данных Delphi каждому полю соответствует собственный объект. Основой объектов полей является класс TFieid, который инкапсулирует основные свойства абстрактного поля, не зависящего от типа данных. От этого базового класса порождены другие класса, обеспечивающие функционирование реальных объектов полей, зависящих от типа данных.

Программист, грамотно использующий возможности полей, может решать существенно более сложные задачи и создавать эффективные и гибкие приложения баз данных.

Эта глава посвящена изучению объектов полей набора данных и приемов работы с ними. В ней рассматриваются следующие вопросы:

- объект поля в наборе данных;
- динамические и статические объекты полей;
- способы использования объектов полей в наборе данных;
- класс TField - основа использования полей в наборах данных;
- типы объектов полей и типы данных;
- ограничения.

Объекты полей

Объекты полей инкапсулируют свойства и методы полей различных типов данных. Они функционируют совместно с набором данных и очень тесно связаны с ним. Например, для того чтобы получить значения полей из текущей записи набора данных, разработчик должен создать примерно такой код:

```
Editl.Text := Tablel.Fields[0].AsString;
```

Свойство Fields представляет собой индексированный список объектов полей набора данных. Если разработчик не изменяет порядок следования полей в наборе данных, то расположение объектов полей в списке Fields соответствует структуре таблицы базы данных.

Каждый объект полей хранит ряд параметров, определяющих поле. Например, в наборе данных к объекту поля можно обратиться, зная только название поля:

```
Editl.Text := Tablel.FieldByName('SomeField1).AsString;
```

Для того чтобы присвоить значение полю в текущей записи, можно воспользоваться приведенными выше способами или, если тип данных поля неизвестен, свойством Fieidvalues:

```
Tablel.FieldValues['SomeField'] := Editl.Text;
```

Знание имени поля дает самый простой способ обращения к текущему значению поля:

```
Tablel['SomeField'] := Editl.Text;
Editl.Text := Tablel['SomeField'];
```

**Примечание**

*При присваивании значений полям набора данных необходимо контролировать состояние, в котором находится набор данных*.

В основе классов, описывающих иерархию типизированных полей, лежит класс TField. От него порождены другие классы, обеспечивающие работу целых групп полей, объединенных по типам данных.

Что же такое объект поля, и какие возможности он предоставляет разработчику?

Во-первых, назначение класса TField, как базового класса поля, заключается в умении взаимодействовать с компонентом отображения данных для обеспечения правильной визуализации данных. Например, объект поля хранит способ выравнивания, параметры шрифта, текст заголовка и т. д.

Во-вторых, с точки зрения набора данных объект поля является хранилищем текущего значения этого поля (а не всего столбца данных, как это можно себе представить по названию).

Компоненты отображения данных при работе с набором данных взаимодействуют именно с полями. Например, колонки компонента TDBCrid при отсутствии дополнительных настроек соответствуют расположению объектов полей в связанном наборе данных.

Статические и динамические поля

В Delphi предусмотрено два способа создания объектов полей.

Динамические поля используются программой в случае, если разработчик не создал для них объекты явным образом на этапе разработки. Каждый не заданный явно объект поля автоматически создается при открытии набора данных в соответствии со структурой связанной таблицы БД. Любой объект поля является прямым наследником класса TField, а его конкретный тип зависит от типа данных поля таблицы. При этом свойства динамического поля устанавливаются в соответствии с параметрами поля таблицы базы данных.

Компонент набора данных после подключения к таблице БД без дополнительных настроек использует только динамические поля. К свойствам и методам динамических полей можно обратиться программно, для этого следует использовать индексированное свойство Fields компонента доступа к данным, которое объединяет все поля набора данных (см. выше) или метод FieldByName.

Динамические поля используются в случаях, когда заданные характеристики полей в таблице базы данных полностью удовлетворяют разработчика и нет необходимости рассматривать какое-либо поле вне набора данных.

Статические поля создаются программистом на этапе разработки, их свойства доступны в Инспекторе объектов, а их названия можно выбрать из списка объектов активной формы в верхней части Инспектора объектов. Название статического объекта поля обычно складывается из названий таблицы и поля, например ordersCUSTNO.

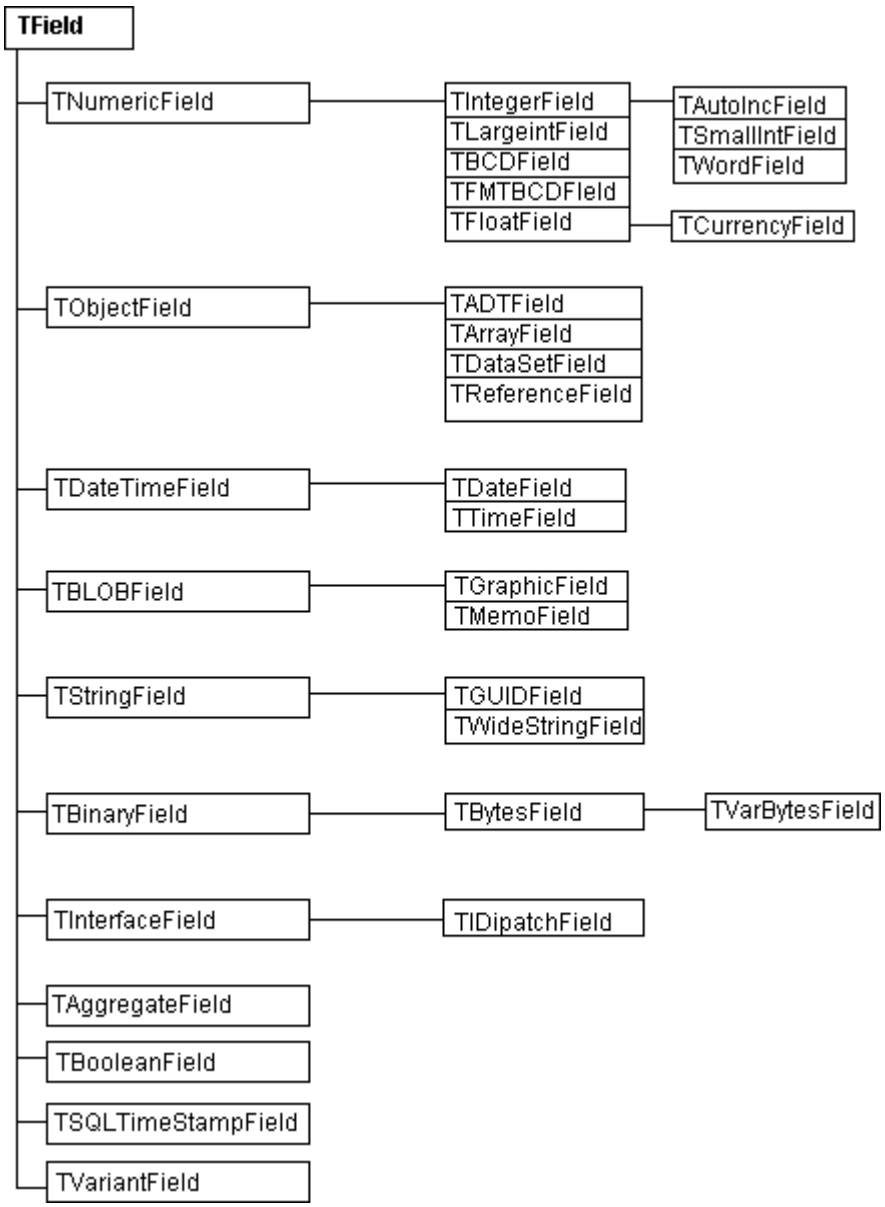


Рисунок 3.1. Иерархия классов полей

Создаются статические объекты полей при помощи специализированного Редактора полей, который вызывается двойным щелчком на компоненте набора данных на форме или командой **Fields Editor** из всплывающего меню этого компонента.

Редактор полей представляет собой простой список уже созданных статических полей. Все управление осуществляется командами из всплывающего меню. В верхней части окна Редактора расположены кнопки навигатора для перемещения по набору данных, которые активны только при открытом наборе данных. Если набор данных имеет агрегатные поля данных, то они размещаются в отдельном списке в нижней части окна Редактора полей (рис. 3.2).

Добавить к списку статических полей новое поле, существующее в таблице БД, можно при помощи команды **Add fields** из всплывающего меню Редактора. Удаление элемента из списка осуществляется клавишей <Delete>. Перетаскиванием элементов списка при помощи мыши можно изменить их

расположение. Таким образом можно создавать произвольные комбинации статических полей.

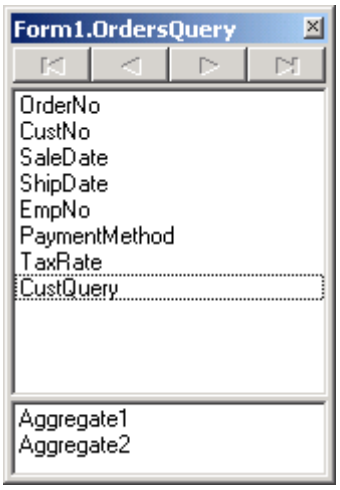


Рисунок 3.2. Редактор полей с отдельным списком агрегатных полей

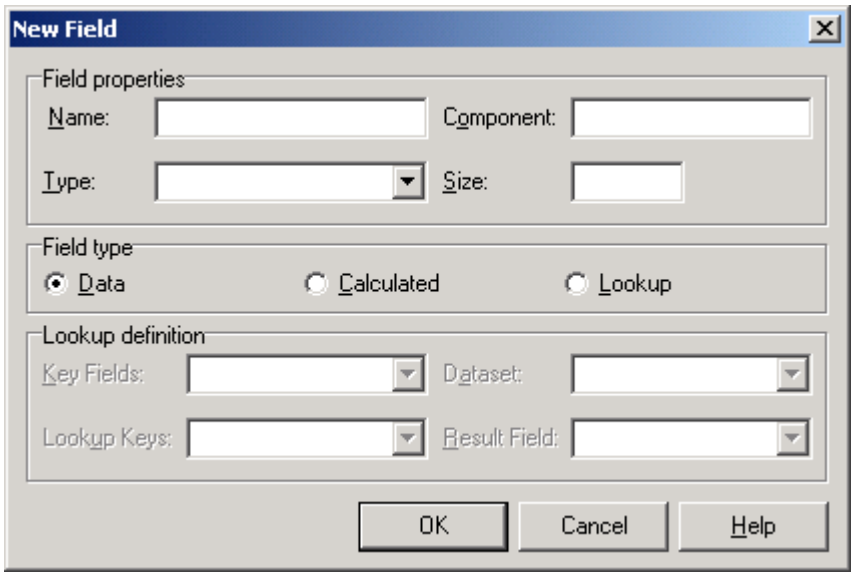


Рисунок 3.3. Диалог создания нового статического поля Редактора полей набора данных

Команда **New field** из всплывающего меню Редактора полей позволяет создать статическое поле, которое реально не существует в структуре данных таблицы (рис. 3.3). Для выбора типа поля используется группа радиокнопок **Field Type:**

- **Data** — поле данных;
- **Calculated** — вычисляемое поле;
- **Lookup** — поле синхронного просмотра.

Реже создаются поля данных, которые обязательно должны базироваться на реальных полях таблицы. Если для такого объекта соответствующее поле в таблице будет удалено или его тип данных будет изменен, то при открытии набора данных генерируется исключительная ситуация.

**Примечание**

*Для клиентских наборов данных многоуровневых приложений диалог создания нового поля позволяет выбрать два дополнительных типа поля — это агрегатные поля (радиокнопка Aggregate) и внутренние вычисляемые поля (радиокнопка InternalCalc).*

**Класс TField**

Как уже говорилось выше, в большой иерархии классов для полей различных типов данных класс TField является базовым (см. рис. 3.1), он инкапсулирует свойства и методы абстрактного поля данных. Именно от него происходят все классы типизированных полей. В реальной работе класс TField не используется, но его значение трудно переоценить. Практически все основные свойства классов типизированных полей унаследованы от класса TField без каких-либо изменений, а дополнительные свойства и методы обеспечивают работу конкретного типа данных.

Что касается методов-обработчиков событий, то четыре метода, определенные в классе TField, наследуются всеми потомками без изменения и дополнения.

Ниже приведены свойства и методы класса TField. Имя объекта содержит свойство

property Name: TComponentName;

При создании объекта поля на этапе разработки имя объекта складывается из имени соответствующего компонента набора данных и имени поля.

Свойство

property FieldName: String;

возвращает имя поля таблицы базы данных. Свойство

property FullName: string;

используется, если текущее поле является дочерним для другого поля. В этом случае свойство содержит имена всех родительских полей.

Название поля в таблице базы данных содержится в свойстве

property Origin: String;

Свойство

property FieldNo: Integer;

возвращает исходный порядковый номер поля в наборе данных. Если объекты полей являются статическими, их фактический порядок может быть изменен в Редакторе полей.

Свойство

property Index: Integer;

содержит индекс объекта поля в списке Fields.

Функциональное назначение поля определяется свойством

type TFieldKind = (fkData, fkCalculated, fkLookup, fklnternalCalc,

fkAggregate);

property FieldKind: TFieldKind;

В большинстве случаев его значение определяется автоматически в момент создания объекта поля. Да и впоследствии вряд ли возникнет необходимость сделать реальное поле данных вычисляемым. Обычно попытка изменить значение

свойства FieldKind вызывает ошибку. Рассмотрим возможные значения этого свойства:

- fkData — поле данных;
- fkCalculated — вычисляемое поле;
- fkLookup — поле синхронного просмотра;
- fklnternalCalc — внутреннее вычисляемое поле;
- fkAggregate — агрегатное поле.

Если поле является вычисляемым, свойство

property Calculated: Boolean;

принимает значение True.

На связанный набор данных указывает свойство

property DataSet: TDataSet;

которое при создании объекта средствами среды разработки заполняется автоматически.

Свойство

property DataType: TFieldType;

возвращает тип данных поля, а свойство

property DataSize: Integer;

содержит объем памяти, необходимый для хранения значения поля.

Одной из важнейших задач класса TField является обеспечение доступа к текущему значению поля. В этом случае класс взаимодействует с буфером текущей записи набора данных, а значение можно получить при помощи нескольких свойств.

Свойство

property Value: Variant

всегда содержит значение, которое сохранено после последнего выполнения метода Post набора данных:

```
with Tablel do begin Open;
while Not EOF do begin
if Fields[0].Value > 10
then Fields[1].Value := Fields[1].Value*2;
 Next;
end;
Close;
end;
```

В этом примере при помощи метода Next осуществляется перебор всех записей набора данных. Если значение первого поля больше 10, то значение второго поля удваивается. Для этого применяется свойство value объектов полей набора данных.

Однако из-за использования вариантов свойство value является относительно медленным. И для преобразования текущего значения поля к необходимому виду можно применять целую группу быстрых свойств AS ..., которые содержат значение в определенном типе данных. Чаще всего используется свойство Asstring, например, оно может применяться для представления числовых значений полей в элементах управления:

Edit1.Text := Tablel.Fields[0].AsString;

**Примечание**

*При работе со статическими объектами полей при передаче значений желательно использовать свойства из группы AS ..., т. к. неявное задание типа свойством Value может привести к ошибке преобразования данных типа Variant.*

Если свойство

property CanModify: Boolean;

имеет значение False, значение поля нельзя редактировать. Однако это свойство является только средством для определения возможности редактирования.

Свойство

property Readonly: Boolean;

позволяет запретить редактирование (Readonly := True) или разрешить его (Readonly := False).

Большая группа свойств отвечает за представление и форматирование значения поля.

Свойство

property DisplayText: String;

содержит значение поля в строковом формате до начала редактирования. Свойство

property Text: String;

предназначено для использования компонентами отображения данных при редактировании. Поэтому эти два свойства могут иметь разные значения в случае, если значение поля в строковом формате при редактировании и просмотре различно. У классов-наследников TField для этого достаточно задать шаблон отображения данных для поля (свойство Display/Format) и шаблон редактирования данных (свойство EditFormat). Например, вещественное число при просмотре может иметь разделители тысяч, а при редактировании нет. В этом случае рассматриваемые свойства будут иметь следующий вид:

DisplayText = ' 1 452,32'

Text = 4452,32'

Свойства Text и DisplayText влияют на использование метода-обработчика onGetText. Если параметр DisplayText имеет значение True, то параметр Text содержит значение свойства DisplayText, в противном случае в метод передается значение поля в строковом формате.

Если поле не имеет значения, то при помощи свойства DefaultExpression можно задать некоторое постоянное значение, которое будет появляться в компоненте отображения данных при пустом поле. Если постоянное значение содержит какие-либо символы кроме цифр, то все выражение нужно обязательно брать в кавычки.

В случае возникновения исключительных ситуаций во время использования поля генерируется соответствующее сообщение, в котором в качестве имени поля применяется значение свойства DisplayName. Если задано свойство DispiayLabel, то DisplayName приравнивается к нему, в противном случае для задания свойства

DisplayName используется свойство FieldName. Другим способом задать значение свойства DisplayName невозможно.

Свойство

property DisplayWidth: Integer;

определяет число символов для отображения значений поля в визуальных компонентах отображения данных.

Свойство

property Visible: Boolean;

отвечает за видимость поля в визуальных компонентах отображения данных. При этом компоненты, отображающие одно поле, перестают показывать его значения, а компоненты типа TDBGrid не отображают колонки, связанные с полем.

**Примечание**

*Еще несколько групп свойств класса TField, а также его методы-обработчики рассматриваются ниже в этой главе.*

Виды полей

Теперь рассмотрим классификацию полей набора данных в зависимости от их функционального назначения. Самыми распространенными полями являются поля данных, базирующиеся на реальных полях таблицы БД. Свойства объектов таких полей устанавливаются в соответствии с параметрами полей таблицы БД.

Кроме этого, в практике программирования часто применяются поля синхронного просмотра и вычисляемые поля.

Процесс создания всех типов полей набора данных практически не отличается (см. выше). Тем не менее такое разнообразие позволяет успешно решать самые сложные задачи программирования приложений БД.

Ниже мы рассмотрим только поля синхронного просмотра и вычисляемые поля, т. к. поля данных не содержат каких-либо существенных особенностей в применении.

С точки зрения набора данных большой разницы между этими двумя видами полей нет. Однако значения для всех полей синхронного просмотра рассчитываются раньше, чем для вычисляемых полей. Поэтому вы можете использовать поля синхронного просмотра в выражениях вычисляемых полей и не можете сделать наоборот.

**Поля синхронного просмотра**

При создании для исходного набора данных нового поля синхронного просмотра необходимо использовать перечисленные ниже свойства.

- Свойство

property LookupDataSet: TDataSet;

задает набор данных синхронного просмотра.

- Свойство

property LookupResultField: String;

представляет поле синхронного просмотра из набора данных LookupDataSet, данные из которого будут появляться в созданном поле.

- Свойство

property LookupKeyFields: String;

содержит поле (или поля) из набора данных синхронного просмотра, по значению которого выбирается значение из поля LookupResultField.

• Свойство

property KeyFields: String;

определяет поле (или поля) из исходного набора данных, для которого создается поле синхронного просмотра.

Для просмотра данных из поля синхронного просмотра можно использовать любые компоненты отображения данных, но естественно будет применить специальные компоненты синхронного просмотра, о которых рассказывается в гл. 15.

Кроме этого, очень удобно использовать поля синхронного просмотра в компоненте TDBGrid. Если такое поле связать с одной из колонок компонента, то для него автоматически заполняется список синхронного просмотра. Его элементы хранятся в свойстве PickList, которое имеется в любой колонке. Теперь пользователю достаточно выбрать нужную колонку в сетке и, щелкнув на появившейся в текущей ячейке кнопке, получить возможные значения для замены. Одновременно с изменением поля синхронного просмотра изменяется и ключевое поле (свойство KeyFields) исходного набора данных.

**Примечание**

*При использовании полей синхронного просмотра в компоненте TDBGrid открывать набор данных синхронного просмотра необязательно. При этом свойство LookupCache, о котором речь пойдет ниже, обязательно должно иметь значение False.*

Для идентификации полей синхронного просмотра можно использовать булевское свойство Lookup базового класса TField, которое принимает истинное значение для таких полей.

Свойство

property LookupCache: Boolean;

определяет режим использования специального буфера значений синхронного просмотра. Если это свойство истинно, то буфер работает.

Буфер основан на свойстве LookupList. При открытии исходного набора данных каждое поле синхронного просмотра получает свое значение, одновременно с этим, в соответствии со всеми имеющимися в исходном наборе данных значениями ключевого поля, заполняется и буфер синхронного просмотра. Впоследствии, при перемещении на другую запись, значение синхронного просмотра берется не из набора данных синхронного просмотра, а из буфера. Этот механизм при небольших объемах значений синхронного просмотра позволяет увеличить скорость работы с исходным набором данных, особенно в режиме удаленного доступа при низкоскоростных сетях.

При изменениях в наборе данных синхронного просмотра можно использовать метод

procedure RefreshLookupList;

который обновляет текущее значение поля и список значений в буфере.

Специально для разработчиков в базовый класс TField включено свойство

property Offset: Integer;

которое возвращает размер буфера в байтах.

Для создания поля синхронного просмотра удобнее всего воспользоваться Редактором полей компонента доступа к данным. После выбора команды **New field** из всплывающего меню в одноименном диалоге (см. рис. 13.3), помимо обычных действий, соответствующих созданию поля данных, необходимо задать значения свойств в группе **Lookup definition.** Элементы управления группы становятся доступны после выбора типа поля (радиокнопка **Lookup** в группе **Field type).** Группа **Lookup definition** включает следующие элементы:

- в списке **Dataset** представлены все доступные в модуле наборы данных, из которых нужно выбрать набор данных синхронного просмотра (свойство Lookup DataSet);
- список **Result Field** позволяет выбрать поле синхронного просмотра (свойство LookupResultField);
- список **Lookup Keys** задает ключевое поле в наборе данных синхронного **Просмотра** (свойство LookupKeyFields);
- список **Key Fields** определяет ключевое поле исходного набора данных (свойство KeyFields).

**Вычисляемые поля**

Вычисляемые поля существенно облегчают разработку приложений баз данных, т. к. позволяют получать новые данные на основе существующих, не изменяя при этом структуру таблиц БД. Выражения для получения значений вычисляемых полей разработчик должен разместить в методе-обработчике OnCalcFields набора данных. Здесь можно использовать любые арифметические, логические операции и функции, любые операторы языка, свойства и методы любых компонентов, в том числе запросы SQL:

```
procedure TForml.TablelCalcFields(DataSet: TDataSet)
; begin
with Tablel do
TabielCalcFieldl.Value := Fields[0].Value + Fields[1].Value;
 with Queryl do
begin
Params[0].AsInteger := Tablel.Fields[0].Aslnteger;
 Open;
TabielCalcFieldl.Value := Fields[0].AsString;
Close;
 end;
 end;
```

Метод OnCalcFields выполняется при открытии набора данных, при переходе в режим редактирования, при передаче фокуса между компонентами отображения данных или колонок сетки, при удалении записи. Но для этого нужно, чтобы свойство AutoCaicFields набора данных было равно значению True.

**Примечание**

*Необходимо учитывать, что сложные вычисляемые поля могут существенно замедлить работу набора данных (особенно при использовании при этом запросов SQL). Кроме того, в процессе редактирования набора данных (при*

*изменении значения поля, сохранении изменений и переходе на следующую запись) вычисляемые поля рассчитываются несколько раз подряд. Для уменьшения числа автоматических обращений к методу OnCalcFields нужно использовать свойство AutoCaicFieids := False.*

Для создания вычисляемого поля достаточно в диалоге создания нового поля Редактора полей в качестве типа поля задать "вычисляемое", в остальном процесс совпадает с созданием поля данных.

В выражениях вычисляемых полей можно использовать другие вычисляемые поля, но они обязательно должны быть определены в методе OnCalcFields до этого**.**

Вычисляемые поля нельзя использовать при фильтрации набора данных при помощи метода-обработчика onFilterRecord, т. к. он вызывается до метода-обработчика OnCalcFields, а вычисляемые поля не сохраняются.

**Внутренние вычисляемые поля**

Помимо простых вычисляемых полей существуют внутренние вычисляемые поля (FieidKind = fkinternaicaic). Они используются в клиентских наборах данных (компоненты TCiientDataSet) и отличаются тем, что их значения сохраняются в наборе данных.

Внутренние вычисляемые поля могут быть использованы для фильтрации методом - обработчиком OnFilterRecord.

**Агрегатные поля**

Агрегатные поля предназначены для выполнения вычислительных операций со значениями полей набора данных с использованием агрегатных функций SQL. К таким функциям относятся:

- AVG — вычисляет среднее значение;
- COUNT — возвращает число записей;
- MIN — вычисляет минимальное значение;
- MAX — вычисляет максимальное значение;
- SUM — вычисляет сумму.

Агрегатные поля не входят в структуру полей набора данных, т. к. агрегатные функции подразумевают объединение записей таблицы для получения результата. Следовательно, значение агрегатного поля нельзя связать с какой-то одной записью, оно относится ко всем или группе записей.

Использование агрегатных полей возможно только в компоненте TClientDataSet и его аналогах, т. к. он обеспечивает кэширование данных, необходимое для проведения вычислений.

Агрегатные поля не отображаются вместе со всеми полями в компонентах TDBGrid, в Редакторе полей они расположены в отдельном списке, а их свойство index (см. выше) всегда имеет значение - 1. Для представления значения агрегатного поля можно воспользоваться одним из компонентов отображения данных, который визуализирует значение одного поля (например, TDBText или TDBEdit), или свойствами самого поля:

Labell.Caption := MyDataSetAGGRFIELDl.AsString;

Для создания агрегатного поля необходимо использовать команду **New field** из всплывающего меню Редактора полей.

Для представления агрегатных полей имеется специальный класс

TAggregateField.

Его свойство

property Expression: string;

задает вычисляемое выражение.

В его состав могут входить агрегатные функции, имена полей набора данных и простейшие арифметические операции:

SUM(Pirce*ItemCount) - SUM(Balance)

Вычисление значения проводится только для тех агрегатных полей, свойство

property Active: Boolean;

которых имеет значение True.

Вычисление включенных свойством Active агрегатных полей выполняется только в том случае, если булевское свойство AggregatesActive клиентского компонента набора данных имеет значение True.

По умолчанию экземпляр класса TAggregateFieid создается со свойством

Visible = False.

Свойство

property GroupingLevel: Integer;

задает уровень группировки полей набора данных при вычислении. При значении 0 расчет проводится для всех записей набора данных. При значении 1 записи группируются по первому полю набора данных и расчет осуществляется для каждой группы. При значении 2 записи разбиваются на группы по первому и второму полям и т. д.

Однако группировка по уровням выше нулевого возможна, только если в наборе данных используется индекс по группирующим полям. Например, если свойство GroupingLevel = 2 и набор данных начинаются с полей CustNo и orderNo, в свойстве indexName компонента набора данных и свойстве

property IndexName: String;

объекта агрегатного поля должно быть имя индекса, включающего оба эти поля.

По причине необходимости подключения индексов, уровень группировки выше нулевого возможен только в табличных компонентах.

**Объектные поля**

Наряду с обычными типами данных (строковым, целочисленным и т. д.), при работе с полями набора данных можно использовать более сложные типы, представляющие собой совокупность более простых типов.

В Delphi существуют- четыре класса объектных полей. Это — TADTField, TArrayField, TDataSetField, TReferenceField. Их общим Предком является класс TObjectField. Классы TADTFieid, TArrayField обеспечивают доступ к набору дочерних полей одного типа из родительского. Эти типы полей можно использовать, если сервер БД поддерживает объектные типы и соответствующие поля имеются в наборе данных. Поэтому объектные поля можно создавать статически и динамически, так же, как и простые поля.

Для доступа к дочерним полям в этих классах имеются свойства:

- property Fields: TFields;

которое представляет собой индексированный список объектов дочерних полей;

- property FieldValues[Index: Integer]: Variant;

которое содержит значения дочерних полей;

- property FieldCount: Integer;

которое возвращает количество дочерних полей.

Классы TDataSetField и TReferenceField предоставляют доступ к данным из связанных наборов данных.

Ссылка на используемый набор данных задается свойством

property DataSet: TDataSet;

**Типы данных**

В среде разработки Delphi можно создавать приложения для работы с самыми разными базами данных. Такая универсальность подразумевает необходимость применения средств, которые бы обеспечили возможность работы со многими типами данных, используемыми в этих базах данных.

Естественно, что существует большая группа типов данных, конкретная реализация которых практически не отличается от платформы к платформе. Это, например, строки, символы, целые и вещественные числа и т. д.

Есть типы данных, которые реализованы далеко не на каждой платформе. Есть, наконец, просто уникальные типы данных.

Для удовлетворения потребностей разработчиков в Delphi применен следующий способ работы с типами данных.

Тип данных однозначно связан с конкретным полем таблицы базы данных. Без этого поля само понятие типа данных не имеет практического смысла. В Delphi свойства абстрактного поля инкапсулирует класс TField, который не имеет заранее определенного типа данных. Уже от этого класса порождено целое семейство классов для типизированных полей (см. рис. 13.1), каждый из которых умеет обращаться со своим типом данных.

**Примечание**

*В классе TField имеется свойство DataType, которое отвечает за тип данных, но оно не может быть изменено.*

Весь список доступных типов данных содержится в типе TFiledType:

type TFieldType = (ftUnknown, ftString, ftSmallint, ftlnteger, ftWord, ftBoolean, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftBytes, ftVarBytes, ftAutoInc, ftBlob, ftMemo, ftGraphic, ftFmtMemo, ftParadoxOle, ftDBaseOle, ftTypedBinary, ftCursor, ftFixedChar, ftWideString, ftLargeint, ftADT, ftArray, ftReference, ftDataSet, ftOraBlob, ftOraClob, ftVariant, ftlnterface, ftlDispatch, ftGuid, ftTimeStamp, ftFMTBcd);

В табл. 3.1 рассматриваются все типы данных, которые можно использовать при разработке приложений для работы с базами данных.

**Примечание**

*В Delphi тип данных BCD напрямую не поддерживается. Его использование обеспечивает денежный тип (ftcurrency). Поэтому точность BCD ограничена 20 цифрами после запятой. Для устранения этого ограничения используется тип FMTBcd, который обладает требуемой точностью.*

Таблица 3.1. Типы данных

| **Тип** | **Класс** | **Описание** |
|---|---|---|
| Неизвестный (ftUnknown) | | Неопределенный тип данных |
| Строковый (ftString) | TStringField | Строка длиной до 8192 символов |
| Целый короткий (ftSmallint) | TSmalllntField | 16-битное целое в диапазоне от -32 768 до 32 767 |
| Целый (ftlnteger) | TIntegerField | 32-битное целое в диапазоне от -2 147 483 648 до 2 147 483 647 |
| Целый положительный (ftWord) | TWordField | 1 6-битное целое в диапазоне от 0 до 65535 |
| Логический (ftBoolean) | TBooleanField | Значения True и False |
| Вещественный (ftFloat) | TFloatField | Вещественные положительные и отрицательные числа с точностью 15 цифр после запятой в диапазоне от 5,0x1 0"$^{324}$ до 1,7x1 0$^{308}$ |
| Денежный (ftCurrency) | TCurrencyField | Вещественные положительные и отрицательные числа с точностью 15 цифр после запятой в диапазоне от 5,0x1 0"$^{324}$ до 1,7x1 0$^{308}$. Дополнительно вставляется символ валюты |
| Десятичный с двоичным кодированием (ftBCD) | TBCDField | Вещественные числа с повышенной точностью (до 4 знаков перед запятой и до 20 знаков после запятой). Могут храниться в двоичном и десятичном форматах |
| Дата (ftDate) | TDateField | Дата |
| Время (ftTime) | TDateTimeField | Время |
| Календарный (ftDateTime) | TDateTimeField | Комбинированный формат с одновременным хранением даты и времени |

| | | |
|---|---|---|
| Фиксированный буфер (ftBytes) | TBytesField | Набор байтов фиксированного размера. Для работы с этим типом требуется выделять и освобождать память (методы GetMem И FreeMem) |
| Переменный буфер (ftVarBytes) | TVarBytes Field | Набор байтов переменного размера. Текущий размер буфера хранится в первых двух байтах. Для работы с этим типом требуется выделять и освобождать память (методы GetMem И FreeMem) |
| Автоинкрементный (ftAutoInc) | TAutoIncField | Значение поля в каждой новой записи автоматически увеличивается на 1 . Целое число в диапазоне от -2 147 483 648 до 2 147 483 647. Применяется для обеспечения уникальности значений ключей |
| BLOB (ftBlob) | TBLOBField | Большой двоичный массив. Используется для хранения любых данных, которые можно преобразовать в цифровой массив (Memo, Graphic). В базах данных такие данные хранятся в отдельных файлах, а поле содержит лишь ссылки на них |
| Memo (ftMemo) | TMemoField | Набор строк произвольной длины |
| Графический (ftGraphic) | TGraphicField | Формат для хранения изображений |
| Форматированный Memo (ftFmtMemo) | | Форматированный набор строк произвольной длины |
| OLE Paradox (ftParadoxOle) | | Поле OLE для таблиц Paradox |
| OLE dBASE (ftDBaseOle) | | Поле OLE для таблиц dBASE |
| Типизированный двоичный (ftTypedBinary) | | Типизированный двоичный |
| Курсор Oracle (ftCursor) | | Курсор для хранимых процедур сервера Oracle |
| Фиксированный символьный (ftFixedChar) | TStringField | Строка символов с нулевым символом в конце |

| | | |
|---|---|---|
| Расширенный строковый (ftWideString) | | Динамически выделяемая строка 16-битных символов в кодировке Unicode |
| Целый большой (ftLargeint) | TLargelntField | 64-битное целое число |
| Абстрактный (ftADT) | TADTField | Произвольный тип данных, создаваемый пользователем на сервере БД и используемый в приложении |
| Массив (ftArray) | TArrayField | Массив полей любого типа, кроме TarrayField |
| Ссылочный (ftReference) | TReferenceField | Указатель на объект, содержащийся в другой таблице |
| Набор данных (ftDataSet) | TDataSetField | Содержит набор данных, интегрированный в текущий набор данных |
| BLOB Oracle 8 (ftOraBlob) | | Тип BLOB для сервера Oracle 8 |
| CLOB Oracle 8 (ftOraClob) | | Тип CLOB для сервера Oracle 8 |
| Вариант (ftVariant) | TVariantField | Вариант |
| Интерфейс (ftlnterface) | TInterfaceField | Ссылка на интерфейс (потомок от lUnknown) |
| Ссылка на интерфейс IDispatch (ftlDispatch) | TIDispatchField | Ссылка на интерфейс (потомок от IDispatch) |
| Глобальный идентификатор (ftGuid) | TGuidField | Глобальный идентификатор GUID |
| Календарный (ftTimeStamp) | | Календарный тип для наборов данных dbExpress |
| Десятичный с двоичным кодированием (ftFMTBcd) | TFMTBCDField | Тип BCD повышенной точности |

Как видно из таблицы, наряду с традиционными типами данных, в Delphi имеются и специальные типы, использование которых значительно расширяет функциональность приложений. В частности, типы ftinterface, ftIDispatch, ftGuid позволяют создавать полноценные приложения БД для COM и OLE DB.

Практически на всех серверах БД пользователь имеет возможность создавать собственные типы данных. Для их использования в приложениях Delphi имеется абстрактный тип данных и класс TADTField. Абстрактный тип может включать любой скалярный тип данных (числа, даты, ссылки, массивы, наборы данных).

Автоинкрементный тип данных давно используется в СУБД. Поле автоинкрементного типа для каждой новой записи автоматически увеличивает свое значение на единицу. Тем самым, каждая запись имеет собственный уникальный идентификатор, который очень часто используется в качестве первичного ключа.

Данные типа BLOB (Binary Large OBject) представляют собой двоичные массивы произвольной длины. В самом поле содержится лишь ссылка на отдельный файл базы данных, в котором хранится двоичный массив. Таким образом, поля типа BLOB являются универсальным носителем любых данных, которые имеют скалярную и нескалярную структуру и которые можно преобразовать в двоичное представление.

Тип Memo представляет собой набор строк произвольной длины (его разновидность — форматированный Memo), основан на формате BLOB. Используется при необходимости сохранить текст из компонента тмето или из текстового редактора.

Графический тип данных используется для хранения в базе данных изображений, основан на формате BLOB. Поле TGraphicFieid непосредственно взаимодействует с компонентом отображения данных (например TDBimage). Изображения должны храниться в формате BMP.

Типы данных ParadoxOle и dBaseOle разработаны специально для использования возможностей СУБД Paradox и dBASE по работе с данными OLE. В Delphi эти типы данных основаны на формате BLOB.

Специально для работы с сервером Oracle 8 предназначены типы CLOB и BLOB.

Тип ftArray организует массив из данных любой структуры, за исключением таких же массивов. Для каждого элемента массива может создаваться собственный объект TField. Для управления этим механизмом используется свойство SparseArrays в классе TDataSet.

В качестве отдельного поля в набор данных можно включить и любой другой произвольный набор данных. Для этого используется специальный тип данных и класс TDataSetField. Причем каждым полем из интегрированного набора данных тоже можно управлять.

Ссылочный тип данных также использует внешние наборы данных, но в этом случае можно подключать и использовать только отдельные поля.

Ограничения

Контроль вводимых в поля набора данных в Delphi возложен на объекты полей, а не на компоненты отображения данных. Именно в рамках этого вопроса мы рассмотрим работу методов-обработчиков событий базового класса TField.

Перед сохранением значения поля в БД всегда вызывается метод-обработчик onvalidate, при этом автоматически проводится проверка на выполнение задаваемых ограничений и ограничений типа данных. Кроме этого, здесь можно предусмотреть свою дополнительную обработку:

```
procedure TForml.TablelSomeFieldValidate(Sender: TFieid);
begin
if (Sender as TFieid).Value < 0 then
```

```
begin
ShowMessage('Значение поля не может быть отрицательным');
(Sender as TFieid).Value := Null;
end;
end;
```

Различают контроль значения в целом и посимвольный контроль. Метод OnValidate проверяет значение поля целиком. Если при проверке обнаружена ошибка, то выдается сообщение и фокус формы устанавливается на соответствующем компоненте отображения данных.

Если метод OnValidate не вызвал исключительной ситуации, то при сохранении значения поля в БД вызывается обработчик события onchange. В принципе, можно предусмотреть операции по контролю данных и в этом методе, но тогда в случае ошибки возникает нежелательная исключительная ситуация, которая может привести к серьезным сбоям в работе приложения.

Проверить текущее значение поля перед его появлением в компоненте отображения данных можно в методе-обработчике OnGetText. Если параметр DisplayText принимает истинное значение, то в параметре Text передается значение свойства DisplayText (значение в строковом формате в таком виде, как оно будет показано в компоненте отображения данных — с символами форматирования). В противном случае в параметре Text передается текущее значение в строковом формате.

**Примечание**

*При использовании метода-обработчика OnGetText на разработчика ложится обязанность самостоятельно предусмотреть передачу значения в компонент отображения данных, в противном случае компонент останется незаполненным.*

В методе-обработчике onSetText можно осуществлять текущий контроль значения в строковом формате в том виде, как оно представлено в компоненте отображения данных. Напомним, что этот обработчик вызывается при каждом изменении свойства Text класса TField.

Рассмотренные методы-обработчики удобнее всего использовать для проверки текущего значения с точки зрения программной логики. Например, чтобы отпускная цена не была ниже закупочной или чтобы остаток не был больше первоначального количества товара в партии. Для проверки правильности самого значения класс TField имеет несколько полезных свойств.

Если на сервере БД задано ограничение на некоторое поле, его можно использовать в приложении Delphi при помощи свойства importedconstraint

Для создания собственного ограничения можно использовать свойство Customconstraint, в котором применяется синтаксис SQL:

```
Value>10
ИЛИ
OutputPrice>InputPricexl.25
```

При возникновении ошибки совсем не лишним будет, если программа выдаст некое осмысленное сообщение, которое поможет пользователю исправить

оплошность. При работе с методами-обработчиками это можно предусмотреть в программном коде.

Для встроенного контроля предусмотрено специальное свойство — GonstraintErrorMessage, которое выводится в виде сообщения при возникновении ошибки. Согласитесь, что это гораздо проще, чем исправлять и перекомпилировать соответствующие файлы ресурсов. Если приложение работает с сервером БД и возникла ошибка ограничения поля, то выводится сообщение, определяемое сервером, а не этим свойством.

Если для поля заданы ограничения, то свойство HasConstraints принимает истинное значение.

Посимвольный контроль данных осуществляется свойством validchars, в котором можно определить допустимые в строковом представлении значения поля символы, и методом isvalidchar, который определяет допустимость использования переданного в параметре символа.

Еще один мощный инструмент контроля данных предоставляет свойство EditMask, которое позволяет создавать шаблоны ввода данных, облегчая тем самым работу пользователя и уменьшая возможность ошибки. Рассмотрим правила создания шаблонов.

Шаблон состоит из трех частей.

Первая часть содержит управляющие символы собственно шаблона. Доступные для создания шаблона символы приведены в табл. 3.2.

Таблица 3.2. Управляющие символы шаблона

| Символ | Описание |
|---|---|
| ! | |
| > | Все символы после этого преобразуются в заглавные |
| < | Все символы после этого преобразуются в строчные |
| <> | Все символы после этого остаются в том регистре, как это было задано пользователем |
| \ | Символ, следующий за этим, считается алфавитным, а не управляющим |
| L | В позиции этого символа обязательно должен находиться только алфавитный символ |
| I | В позиции этого символа может находиться алфавитный символ |
| A | В позиции этого символа обязательно должен находиться алфавитный символ или цифра |
| a | В позиции этого символа может находиться алфавитный символ или цифра |
| C | В позиции этого символа обязательно должен находиться знак препинания |

| | |
|---|---|
| c | В позиции этого символа может находиться знак препинания |
| 0 | В позиции этого символа обязательно должна находиться цифра |
| 9 | В позиции этого символа может находиться цифра |
| # | В позиции этого символа может находиться цифра, плюс или минус |
| : | Символ разделения часов, минут и секунд (зависит от системных установок) |
| / | Символ разделения дней, месяцев, годов (зависит от системных установок) |
| ; | Символ разделения частей шаблона |
| - | Символ автоматического ввода в текст пробела |

В первую часть шаблона можно включать любые алфавитные символы (для создания поясняющих надписей, слов и сокращений), если их нет среди управляющих символов. Также можно использовать в качестве алфавитных и управляющие символы, для этого перед ними нужно помещать символ "\".

Вторая часть состоит из одного символа и определяет, могут ли не арифметические символы быть частью вводимого текста. Если здесь расположен ноль, то можно вводить только цифры, если любой другой символ — можно использовать и алфавитные символы.

В третьей части содержится символ, используемый для обозначения мест, запрещенных для ввода.

Части шаблона разделяются точкой с запятой.

Например, шаблон для ввода телефонного номера выглядит следующим образом:

!\{999\)000-0000;1;_

Резюме

Работа с полями является важным этапом в процессе разработки приложения баз данных. Для этого используются специальные объекты, которые инкапсулируют возможности полей таблицы БД. В Delphi имеется целая иерархия классов, обеспечивающая применение полей самых различных типов. В основе этой иерархии лежит класс TField.

По способу создания объекты полей делятся на статические и динамические.

По функциональным возможностям объекты полей бывают полями данных, вычисляемыми, синхронного просмотра, агрегатными.

Объекты полей играют важную роль в работе наборов данных. С их помощью можно получить доступ к текущим значениям, задать ограничения на вводимые величины и проверить их правильность.

**Форма контроля: тестовые вопросы**

1. Этот компонент отображает текущее значение связанного с ним поля набора данных и позволяет изменить его на любое фиксированное из списка.

a) TDBListBox
b) TDBGrid
c) TDBText
d) TDBRadioGroup
e) TDBCheckBox

2. Этот компонент отображает текущее значение связанного с ним поля набора данных в строке редактирования, при этом значение поля должно совпадать с одним из элементов разворачивающегося списка.
a) TDBComboBox
b) TDBGrid
c) TDBText
d) TDBRadioGroup
e) TDBCheckBox

3. В Delphi существуют- четыре класса объектных полей. Это —
a) TADTField, TArrayField, TDataSetField, TReferenceField
b) TADTField и TDataSetField
c) TDBMemo и TDBRichEdit
d) TDBMemo и TDBRichEdit
e) TADTField TDataSetField

4. Какие компоненты используются для представления и редактирования информации, содержащейся в полях типа Memo
a) TDBMemo и TDBRichEdit
b) TADTField и TDataSetField
c) TADTField, TArrayField, TDataSetField, TReferenceField
d) TDBImage и ArrayField
e) TDBEdit

5. Для просмотра (без редактирования) изображений предназначена компонента
a) TDBImage
b) TDBMemo
c) TADTField TDataSetField
d) TDBRichEdit
e) TArrayField

6. Этот компонент инкапсулирует двумерную таблицу, в которой строки представляют собой записи, а столбцы — поля набора данных.
a) TDBGrid
b) TDBMemo
c) TDBEdit
d) TDBText
e) TArrayField

7. Этот компонент представляет собой статический текст, который отображает текущее значение некоторого поля связанного набора данных.
a) TDBText
b) TDBGrid
c) TDBCheckBox
d) TDBEdit
e) TArrayField

8. Этот компонент представляет собой стандартный однострочный текстовый редактор, в котором отображаются и изменяются данные из поля связанного набора данных.
a) TDBEdit
b) TDBGrid
c) TDBText
d) TDBRichEdit
e) TDBCheckBox

9. Этот компонент представляет собой почти полный аналог обычного флажка (компонент TCheckBox) и предназначен для отображения и редактирования любых данных, которые могут иметь только два значения.
a) TDBCheckBox
b) TDBGrid
c) TDBText
d) TDBRichEdit
e) TArrayField

10. Этот компонент представляет собой стандартную группу переключателей, состояние которых зависит от значений поля связанного набора данных.
a) TDBRadioGroup
b) TDBGrid
c) TDBText
d) TDBRichEdit
e) TDBCheckBox

## Рекомендуемая литература

**Основная литература**

1. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.

2. Майерс Г. Искусство тестирования программ. - М.: Финансы и статистика, 1982.

Фокс Дж. Программное обеспечение и его разработка. - М.: Мир, 1985.

3. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2007 г.

4. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.

**Дополнительная литература**

5. Дарахвелидзе П.Г., Марков Е.П. Программирование в Delphi7. Спб.: БХВ - Санкт-Петербург,2003.-784с.:ил.

6. Смайлова Ә.М. DELPHI ортасында деректер базасын бағдарламалау. Оқу құралы. - Алматы, 2011.

## Тема № 4
## Механизмы управления данными

**План:**

1. Связанные таблицы
2. Поиск данных
3. Фильтры
4. Быстрый переход к помеченным записям
5. Диапазоны

Наряду с описываемыми в предыдущих лекциях свойствами и методами, стандартный набор данных Delphi инкапсулирует ряд дополнительных механизмов, облегчающих управление записями и полями.

К ним относятся такие полезные функции, как быстрое перемещение по записям, поиск нужной записи по значениям полей, дополнительная фильтрация записей набора данных без использования возможностей СУБД и т. д. Большинство этих механизмов применяют в своей работе индексы таблиц БД.

Абстрактные методы, обеспечивающие управление данными, реализованы в базовом классе TDataSet (см. гл. 12). А классы-потомки, в свою очередь, реализуют механизмы управления данными в соответствии с возможностями технологий доступа к данным (см. часть IV).

Все рассматриваемые в этой главе методы управления данными в полном объеме доступны только в компонентах, инкапсулирующих таблицу БД. Это связано с тем, что компоненты запросов SQL и хранимых процедур не обеспечивают полноценное использование индексов.

В этой главе рассматриваются следующие вопросы:

- связанные таблицы;
- методы поиска данных;
- диапазоны;
- быстрая навигация по набору данных;
- фильтрация записей в наборе данных.

Связанные таблицы

В рамках одного проекта таблицы БД можно связывать отношениями "один-ко- многим" и "многие- ко- многим", при этом отношения обязательно устанавливаются между индексированными полями двух таблиц.

При создании отношений в качестве главной таблицы можно использовать любой компонент, инкапсулирующий набор данных. Для задания подчиненной таблицы можно использовать только табличные компоненты (см. гл. 12).

**Отношение "один- ко- многим"**

Для установления отношения "один- ко- многим" в наборе данных предназначены два свойства — Mastersource и MasterFieids, которые задаются для подчиненной таблицы. Набор данных главной таблицы не требует никаких дополнительных настроек и заданная связь будет работать только при перемещениях по записям главной таблицы.

Свойство Mastersource определяет компонент TDataSource, который связан с главной таблицей.

Затем при помощи свойства MasterFields необходимо установить отношения между полями главной и подчиненной таблицы. В нем содержится имя индексированного поля, по которому устанавливается связь. Если таких полей несколько, их имена разделяются точкой с запятой. При этом не все поля, входящие в индекс, обязаны участвовать в создании отношения.

Для задания свойства MasterFields можно использовать Редактор связей полей (Field Link Designer), который вызывается щелчком на кнопке в поле редактирования этого свойства в Инспекторе объектов (рис. 4.1).

Здесь в разворачивающемся списке **Available Indexes** выбирается требуемый индекс для подчиненной таблицы. После этого в списке **Detail Fields** появляются имена всех полей, входящих в этот индекс. В списке **Master Fields** отображаются все поля главной таблицы.

Теперь требуется создать связи между полями. Для этого в левом списке выбирается поле подчиненной таблицы, а затем соответствующее ему поле главной таблицы в правом списке. После этого активизируется кнопка **Add**, щелчок на которой создает отношение по двум полям главной и подчиненной таблиц. Созданная связь отображается в списке **Joined Fields.**

**Примечание**

*После создания связи по индексированным полям данный индекс становится текущим для набора данных. При этом в зависимости от типа СУБД автоматически заполняется свойство indexName или indexFieldNames.*

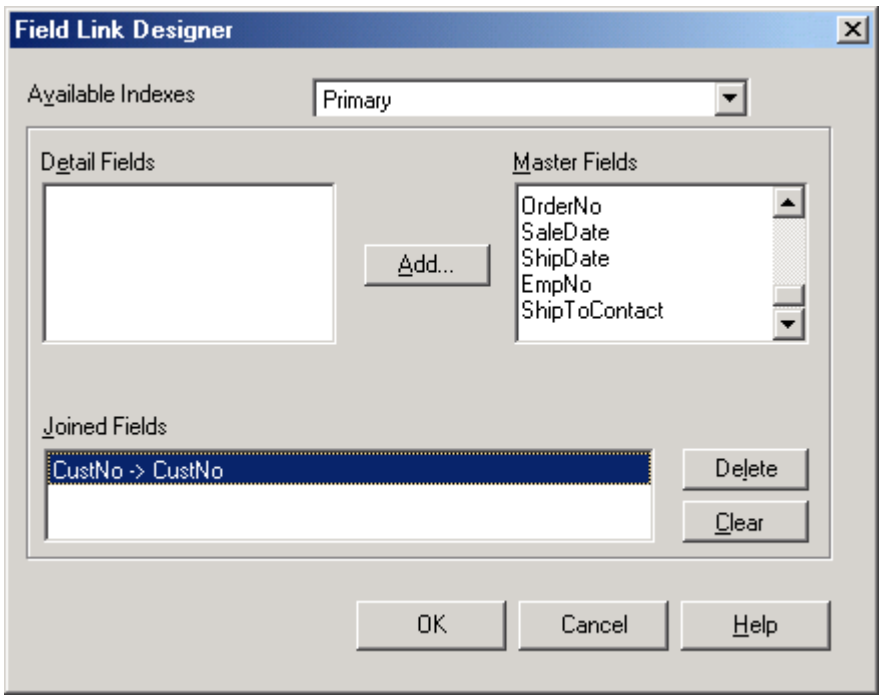


Рисунок 4.1. Редактор связей полей

Уже созданные связи можно удалить. Кнопка **Delete** удаляет выбранную связь, кнопка **Clear** — все связи.

После создания связей между полями отношение "один- ко- многим" считается установленным. Теперь достаточно открыть оба набора данных, чтобы увидеть работу отношения.

В качестве примера рассмотрим проект DemoJoins, в котором связываются таблицы из демонстрационной базы данных DBDEMOS. Для этого использованы компоненты ADO.

Таблица Customers представлена в наборе данных компонента CustTable, она содержит данные о покупателях. Таблица Orders представлена в наборе данных компонента ordTable, она содержит данные о заказах. Таблица Employee представлена в наборе данных компонента EmpTable, она содержат данные о продавцах (рис. 4.2).

**Примечание**

*Приложение DemoJoins не содержит дополнительного исходного кода. Все отношения между таблицами заданы при помощи Инспектора объектов.*

Отношение "один- ко- многим" задано между таблицами покупателей (Customers) и заказов (Orders). Таблица покупателей является главной. Для создания отношения установлены следующие значения свойств компонента ordTable (подчиненная таблица).

Свойство MasterSource должно указывать на компонент custsource, связанный с набором данных CustTable.

Свойство MasterFields указывает на поле custNo таблицы Customers.

В наборе данных OrdTable включен вторичный индекс на основе поля CustNo (indexName = 'CustNo').

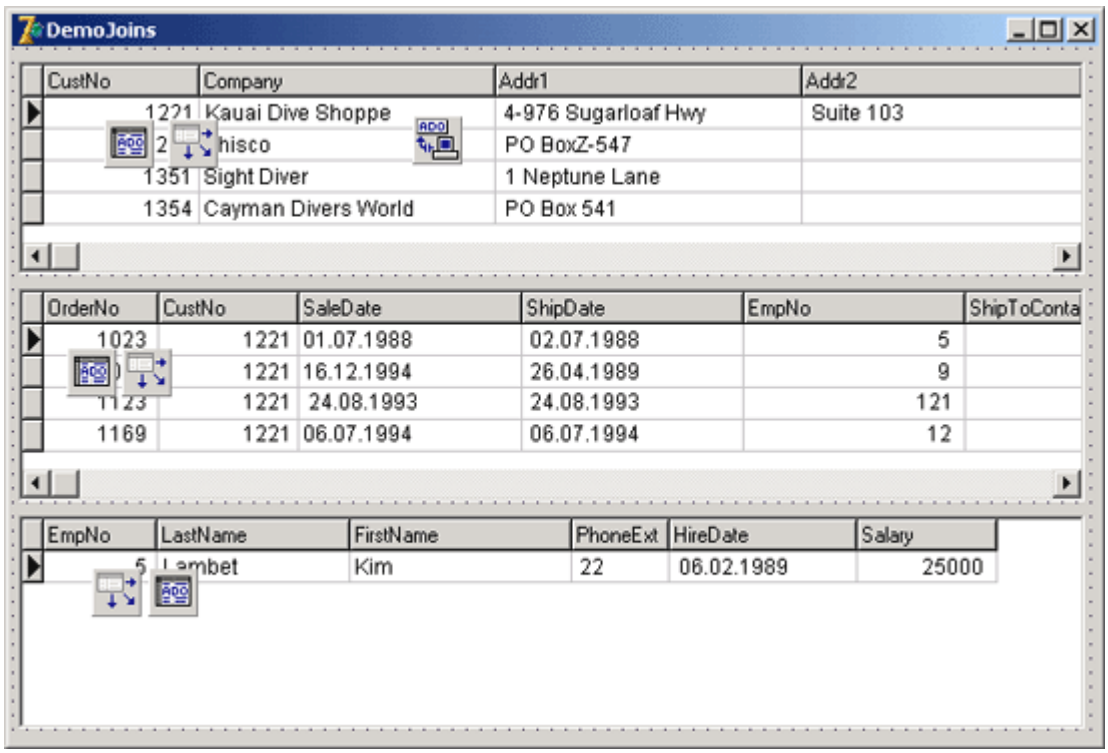


Рисунок 4.2. Главная форма проекта DemoJoins

Таким образом, две таблицы связаны отношением "один- ко- многим" по индексированным полям custNo (номер покупателя). В результате, при перемещении по записям таблицы покупателей, в таблице заказов будут показаны только те заказы, которые относятся к текущему покупателю

**Отношение "многие- ко- многим"**

Отношение "многие- ко- многим" отличается тем, что подчиненная таблица еще раз связывается в качестве главной с другой подчиненной таблицей аналогичной последовательностью действий, как и в отношении "один- ко- многим".

В приложении DemoJoins отношением "многие- ко- многим" связаны таблицы заказов (Orders) и продавцов (Employee). Таблица заказов уже работает в отношении "один- ко- многим" в качестве подчиненной.

В наборе данных EtpTable заданы следующие свойства:

- свойство MasterSource указывает на компонент Empsource;
- свойство MasterFields содержит имя поля EmpNo, по которому осуществляется связь между таблицами. Для подчиненной таблицы поле EmpNo является первичным.

Поиск данных

В наборе данных реализованы два способа поиска записей по заданным значениям полей. Один способ основан на использовании индексов и является более быстрым, но поиск проводится только по индексированным полям. Второй способ применяет специальные методы классов наборов данных и позволяет проводить поиск по любому сочетанию полей, но он более медленный.

**Поиск по индексам**

Для организации индексного поиска к набору данных должен быть подключен индекс (свойства IndexName ИЛИ IndexFieldNames).

Метод FindKey проводит поиск записи по заданным в параметре значениям ключевых полей текущего индекса набора данных. В случае успеха курсор набора данных устанавливается на найденной записи, а метод возвращает значение True, в противном случае — False.

Если индекс состоит из нескольких полей, значения для поиска записываются в виде множества, причем отсутствующие значения приравниваются к Null.

Рассмотрим простейший пример, в котором реализован поиск по вторичному индексу в таблице CUSTOLY.DB демонстрационной базы данных DBDEMOS. Индекс основан на полях Last_Name И First_Name (рис. 4.3).

В компоненте table1, помимо стандартных настроек на таблицу, при помощи свойства IndexName задан и вторичный индекс (его имя Names). Значения для поиска задаются в компонентах Edit1 и Edit2.

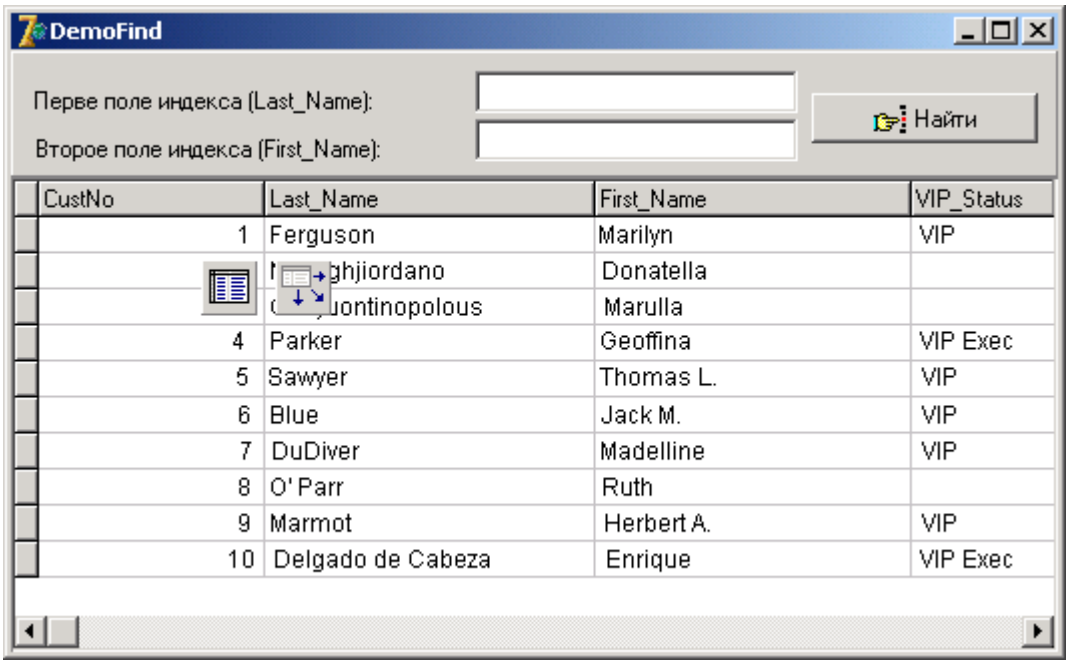


Рисунок 4.3. Главная форма проекта DemoFind

**Листинг 14.1.** Секция Implementation главного модуля Main проекта DemoFind

```
implementation
{$R *.DFM}
procedure TForml.FormShow(Sender: TObject);
begin
try
Cust.Open;
except
on E: EDBEngineError do ShowMessage('Ошибка при открытии таблицы');
end;
```

```
end;
procedure TForml.FormClose(Sender: TObject; var Action: TCloseAction);
begin
Gust.Close;
end;
procedure TForml.FindBtnClick(Sender: TObject);
begin
try
if not Gust.FindKey([Editl.Text, Edit2.Text])
then ShowMessage('Запись не найдена');
except on E: EDatabaseError
do ShowMessage('Ошибка поиска');
end;
end;
end.
```

Набор данных открывается в методе-обработчике FormShow при открытии формы и закрывается в методе-обработчике Formclose. При щелчке на кнопке FindBtn в метод FindKey передаются значения для поиска из компонентов Edit1 И Edit2.

**Поиск в диапазоне**

Индексный поиск можно организовать группой методов, подобно созданию диапазонов. Метод setKey переводит набор данных в состояние dsSetKey, затем должно следовать присваивание ключевым полям значений для поиска. Сам поиск осуществляется методом GotoKey:

```
with Tablel do begin
SetKey;
Fields[0].Value := '428';
GotoKey; end;
```

В случае успеха курсор набора данных устанавливается на найденной записи, а метод возвращает значение True. Вместо этого метода можно применять метод GotoNearest, который в случае неудачного поиска ищет запись, минимально отличающуюся от критерия поиска.

Изменение параметров поиска осуществляется методом EditKey.

**Поиск по произвольным полям**

Для поиска по произвольной выборке полей можно использовать методы Locate и Lookup.

```
function Locate(const KeyFields: string;
const KeyValues: Variant; Options; TLocateOptions): Boolean;
function Lookup(const KeyFields: string;
const KeyValues: Variant; const ResultFields: string): Variant;
```

В метод Locate необходимо передать список полей, по которым будет идти поиск (параметр KeyFields, имена полей разделяются точкой с запятой), их требуемые значения (параметр KeyValues, значения разделяются запятой) и настройки поиска (параметр options). В настройках можно задать опцию loCaseinsensitive, которая отключает проверку на регистр символов, и опцию

loPartialKey, которая включает поиск с минимальными отличиями. В случае успеха поиска курсор набора данных устанавливается на найденной записи, а метод возвращает значение True.

```
Tablel.Locate('Last__Name;First_Name', VarArrayOf(['Editl.Text',
 'Edit2.Text']), []};
```

В метод Lookup передается список полей для поиска (параметр KeyFields, имена полей разделяются точкой с запятой) и их требуемые значения (параметр KeyValues, значения разделяются запятой). В случае успешного поиска функция возвращает массив значений типа вариант для полей, названия которых содержатся в параметре ResultFields.

```
Tablel.Lookup('Last_Name;First_Name',
VarArrayOf(['Editl.Text', 1Edit2.Text']), 'Last_Name;First_Name');
```

Оба эти метода автоматически используют быстрый индексный поиск в случае, если в параметре KeyFields задать поля индекса.

Фильтры

Наиболее эффективным способом отбора записей в набор данных (особенно из больших таблиц) является создание и выполнение соответствующего запроса SQL. Но что делать, если набор данных функционирует на базе табличного компонента? В этом случае на помощь приходит встроенный в набор данных механизм фильтрации данных.

Применение фильтра основано всего на двух основных свойствах и одном вспомогательном. Текст фильтра должен содержаться в свойстве Filter, а свойство Filtered включает и выключает фильтр. Параметры фильтра определяются свойством FilterOptions.

**Примечание**

*Компонент TQuery также может использовать фильтры. Эта возможность подчас позволяет легко и изящно решать довольно сложные проблемы, которые иначе требуют изменения текста запроса или создания нового компонента запроса.*

При использовании фильтра его текст транслируется в синтаксис SQL и передается для выполнения на сервер или через соответствующий драйвер в локальную СУБД.

Фильтры можно создавать двумя способами:

- при помощи свойства Filter создаются довольно простые фильтры, для которых достаточно предоставляемого механизмом фильтрации синтаксиса;
- для создания более сложных фильтров с применением всех возможных средств языка программирования применяется метод-обработчик набора данных OnFilterRecord.

Фильтры можно разделить на статические и динамические.

Статические фильтры создаются во время разработки приложения и могут использовать как свойство Filter, так и метод OnFilterRecord.

Динамические фильтры можно создавать и редактировать во время выполнения приложения, для них используется только свойство Filter.

При создании текста фильтра для свойства Filter используются имена полей соответствующей таблицы БД, а для задания отношений применяются все

операторы сравнения (>, >=, <, <=, =, <>) и логические операторы (AND, OR, NOT):

Fieldl>100 AND Field2=20

Сравнивать между собой два поля нельзя. Следующий фильтр вызовет ошибку при попытке использования:

ItemCount=Balance AND InputPrice>OutputPrice

При создании динамических фильтров можно изменять как выражение фильтра целиком, так и его части. Например, ограничивающее значение для поля можно задавать при помощи элементов управления формы, что позволяет пользователю приложения управлять фильтрацией набора данных:

```
procedure TForml.EditlChange(Sender: TObject);
begin
with Tablel do begin
Filtered := False;
Filter := 'Fieldl>=' + TEdit(Sender).Text; Filtered := True;
end;
end;
```

В фильтрах можно производить отбор по частям строк для строковых полей, для этого используется символ звездочка:

ItemName='A*'

Фильтр начинает работать только после того, как свойству Filtered присваивается истинное значение. Перед изменением текста динамического фильтра или для отключения фильтра свойству Filtered присваивается значение False.

Параметры фильтра определяются свойством FilterOptions:

```
property FiiterOptions: TFilterOptions;
TFilterOption = (foCaselnsensitive, foNoPartialCompare);
TFilterOptions = set of TFilterOption;
```

Параметр foCaselnsensitive, будучи включенным в свойстве, отключает сравнение строковых значений с учетом регистра символов.

Параметр foNoPartialCompare отключает отбор строковых значений по части строки.

Метод-обработчик onFilterRecord имеет следующее объявление:

```
type TFilterRecordEvent = procedure(DataSet: TDataSet; var Accept:
Boolean) of object;
property OnFiiterRecord: TFilterRecordEvent;
```

Если этот метод создан для набора данных, то он вызывается для каждой его записи. Программный код метода должен присваивать параметру Accept истинное или ложное значение. В результате запись передается в набор данных или отсекается:

```
procedure TForml.TablelFilterRecord(DataSet: TDataSet;
 var Accept: Boolean);
begin
Accept := ArchOrdersArchDat.AsString >= DateEditl.Text;
end;
```

Важнейшее преимущество метода onFilterRecord, по сравнению со свойством Filter, заключается в том, что в этом методе-обработчике можно сравнивать поля и производить вычисления над их значениями.

Недостатком метода является недостаточная гибкость, хотя такой фильтр можно модифицировать путем присвоения методу процедурной переменной, содержащей ссылку на новый метод.

Быстрый переход к помеченным записям

Закладки, как инструмент работы с записями набора данных, позволяют осуществлять быстрое перемещение на нужную запись. Набор данных может содержать неограниченное число закладок, каждая из которых представляет собой указатель. Закладку можно создать только для текущей записи набора данных.

При работе с закладками используются три основных метода:

- метод GetBookmark создает новую закладку для текущей записи;
- метод GotoBookmark осуществляет переход к закладке, переданной в параметре;
- метод FreeBookmark удаляет закладку, переданную в параметре.

Кроме этого, можно использовать метод Bookmarkvalid, который проверяет, указывает ли закладка на реально существующую запись. Метод compareBookmark позволяет сравнить между собой две закладки:

```
var Bookmarkl, Bookmark2: TBookmark;
...
if Tablel.CompareBookmark(Bookmarkl, Bookmark2) = 1
 then ShowMessage (' Закладки одинаковы') ;
```

В наборе данных имеется свойство Bookmark, которое содержит название текущей закладки.

Рассмотрим небольшой пример, где право управлять закладками предоставлено пользователю (рис. 4.4). На форме, помимо других элементов управления (среди которых есть компонент TDBGrid), имеются две кнопки. Кнопка startBookmark помечает текущую запись, кнопка stopBookmark переходит к закладке, а затем уничтожает ее.

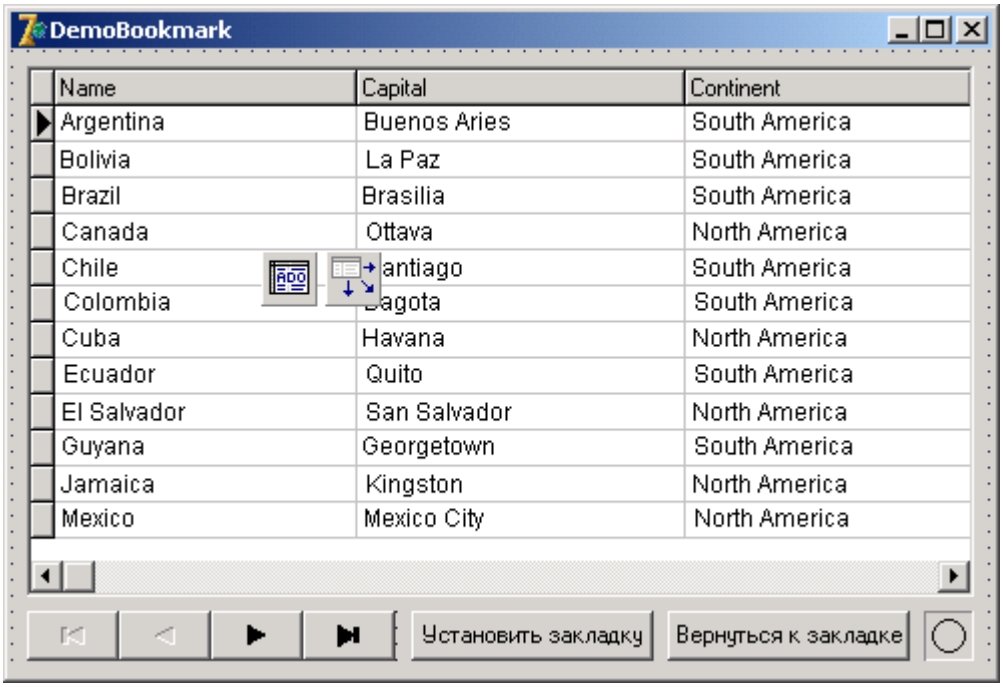


Рисунок 4.4. Главная форма проекта DemoBookmark

**Листинг 14.2.** Пример использования закладок **.**

```
implementation
{$R *.DFM}
var SaveRecPos: TBookMark;
procedure TMainForm.FormShow(Sender: TObject);
 begin
try
Cust.Open;
BookmarkControl.Brush.Color := clBtnFace;
 except
ShowMessage('Ошибка открытия набора данных');
 end;
end;
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
 begin
Cust.Close;
 end;
procedure TMainForm.StartBookmarkClick(Sender: TObject);
begin
if Not Cust.BookmarkValid(SaveRecPos)
then SaveRecPos := Cust.GetBookmark;
BookmarkControl.Brush.Color := cILime
 end;
procedure TMainForm.StopBookmarkClick(Sender: TObject);
 begin
with Cust do begin if Not BookmarkValid(SaveRecPos)
then Exit;
GotoBookmark(SaveRecPos);
```

```
 FreeBookmark(SaveRecPos);
 SaveRecPos := Nil;
end;
BookmarkControl.Brush.Color := clBtnFace;
 end;
end.
```

Использование метода Bookmarkvaiid позволяет корректно переопределять закладку, если она уже установлена, и избежать ошибок при произвольных нажатиях кнопок. Компонент BookmarkControl типа TShape сигнализирует о том, что закладка установлена или удалена.

**Примечание**

*Закладки также используются в компоненте TDBGrid. Он имеет свойство SelectedRows типа TBookmarkList, которое представляет собой список закладок, указывающих на одновременно выделенные записи.*

Диапазоны

В наборе данных, помимо фильтров, имеется еще одно средство отбора записей. Группа методов позволяет на основе использования индексов отбирать в набор данных только те записи, значения индексированных полей которых (для текущего индекса) соответствуют диапазону заданных величин.

Диапазоны работают быстрее фильтров, но менее гибки и не так удобны в работе. При использовании диапазонов набор данных обязательно должен находиться в состоянии dsSetKey (см. ниже).

Для того чтобы включить диапазон, необходимо задать стартовое и конечное значение диапазона для ключевых полей, затем применить созданный диапазон к набору данных. Работающий диапазон можно модифицировать.

**Примечание**

*Все методы работы с диапазонами используют те поля, которые заданы в текущем индексе. Для таблиц Paradox и dBASE это свойство indexName. Для таблиц серверов SQL это свойство indexFieldNames.*

Метод setRangestart переводит набор данных в режим dsSetKey, следующее за этим присваивание ключевым полям значений означает задание начальной границы диапазона.

Метод setRangeEnd переводит набор данных в режим dsSetKey, следующее за этим присваивание ключевым полям значений означает задание конечной границы диапазона.

После этого необходимо использовать метод AppiyRange, который применяет созданный диапазон к набору данных:

```
with Tablel do
begin
SetRangeStart;
Fields[0].Value := '439';
SetRangeEnd;
Fields[1].Value := '522';
AppiyRange;
end;
```

Работающий диапазон можно модифицировать аналогичным образом: после вызова методов EditRangestart и EditRangeEnd необходимо задать новые границы для ключевых полей и снова вызвать метод ApplyRange:

```
with Tablel do
 begin
EditRangeStart;
Fields[0].Value := '500';
EditRangeEnd;
Fields[1].Value := '522';
AppiyRange;
end;
```

Отмена диапазона осуществляется методом CancelRange.

Если индекс содержит несколько полей, то перед вызовом метода AppiyRange необходимо задать значения для всех ключевых полей.

Для одновременного задания верхней и нижней границы диапазона можно использовать метод SetRange.

```
with Tabiel do
 begin
SetRange(['500'], ['522']);
AppiyRange;
end;
```

Тем, какая граница будет у диапазона — открытая или закрытая, управляет свойство KeyExclusive. Если оно имеет значение True, граничные значения в диапазон не включаются, в противном случае — включаются.

Резюме

Разработчик приложений БД в Delphi может использовать ряд полезных механизмов набора данных, которые реализованы для компонентов всех технологий доступа к данным.

К этим механизмам относятся методы быстрого поиска и перехода к найденным записям; связывания наборов данных по индексированным полям; метод дополнительной фильтрации записей набора данных.

**Форма контроля: тестовые вопросы**

1. Какие свойства компонента Table необходимы для отображения данных:
a) Все верно
b) DataBaseName
c) TableName
d) TableType
e) Active

2. Какие свойства компонента DataSource необходимы для отображения данных:
a) Name, DataSet
b) Name, DataBaseName
c) TableName

d) Active
e) все верно

3. Запись данных для компонента DBText:
a) DBText1.DataField=Имя поля
b) DBText1.Caption=’Запись’
c) DBText1.Value=’Запись’
d) DBText1=’Запись’
e) все верно

4. Чем отличается компонент DBEdit, DBMemo от компонента DBText:
a) можно редактировать
b) можно просматривать
c) можно перемещать
d) ничем
e) можно вращать

5. Выберите свойства компонента DBNavigator для перемещения записей в таблице и отображения данных в таблице:
a) Name, DataSource, VisibleButtons
b) Name
c) DataSource
d) VisibleButtons
e) Active

6. Выберите свойства компонента DBGrid для отображения записей в таблице:
a) все верно
b) Name, DataSource, Columns
c) Options.dgTitles
d) Options.dgIndicator
e) Options.ColumnResize, Options.dgColLines, Options.dgRowLines

7. Какие свойства компонента Columns необходимы для оформления таблицы?
a) Все верно
b) FieldName, Width
c) Font, Color
d) Alignment, Title.Caption, Title.Alignment
e) Title.Color, Title.Font

8. Какие свойства для компонента Query необходимы для выполнения запроса:
a) Name, SQL, Active
b) Name
c) SQL
d) Active
e) Font, Color

9. Можно ли редактирование записей в таблице, и с помощью какого свойства:
a) Edit
b) Insert
c) Post
d) Edition
e) Нельзя

10. Можно ли вставить запись в таблицу, и с помощью какого свойства:
a) Insert
b) Edit
c) Post
d) Edition
e) Нельзя

**Рекомендуемая литература**

**Основная литература**

1. Брукс Ф. П. Как проектируются и создаются программные комплексы. М.: Наука, 2009.

2. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.

Фокс Дж. Программное обеспечение и его разработка.-М.: Мир, 1985.

3. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2007 г.

4. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.

**Дополнительная литература**

5. С.Бобровский. Delphi 7. Учебный курс. Издательство «ПИТЕР»,2005г.

6. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».

7. Методические указания для выполнения лабораторных работ по дисциплине

## Тема № 5
## Компоненты отображения данных

**План:**

1. Классификация компонентов отображения данных
2. Табличное представление данных
3. Навигация по набору данных
4. Представление отдельных полей
5. Синхронный просмотр данных
6. Механизм синхронного просмотра
7. Графическое представление данных

До этого момента мы рассмотрели аспекты создания приложений баз данных, касающиеся организации доступа к данным и создания в приложениях наборов данных. Теперь более подробно остановимся на вопросах отображения данных в приложениях (интерфейс приложений).

Отображение данных обеспечивает достаточно представительный набор компонентов VCL Delphi. Многие из них унаследованы от компонентов, инкапсулирующих стандартные элементы управления. Для связи с набором данных эти компоненты используют компонент TDataSource.

Механизмы управления данными реализованы в компонентах наборов данных и активно взаимодействуют с компонентами отображения данных.

В этой главе рассматриваются следующие вопросы:

- использование стандартных компонентов отображения данных;
- навигация по данным;
- механизм синхронного просмотра данных;
- использование графиков для представления данных.

Классификация компонентов отображения данных

Все компоненты отображения данных можно разделить на группы по нескольким критериям (рис. 5.1).

Большинство компонентов предназначены для работы с отдельным полем, т. е. при перемещении по записям набора данных такие компоненты показывают текущие значения только одного поля. Для соединения с набором данных через компонент TDataSource предназначено свойство DataSource. Поле задается свойством DataField.

Компоненты TDBGrid и TDBCtrlGrid обеспечивают просмотр наборов данных целиком или в произвольном сочетании полей. В них присутствует только свойство DataSource.

Особенную роль среди компонентов отображения данных играет компонент TDBNavigator. Он не показывает данные и не предназначен для их редактирования, зато обеспечивает навигацию по набору данных.

Наиболее часто в практике программирования используются компоненты TDBGrid, TDBEdit И TDBNavigator.


Рисунок 5.1. Классификация компонентов отображения данных

Для представления и редактирования информации, содержащейся в полях типа Memo, используются специальные компоненты TDBMemo и TDBRichEdit.

Для просмотра (без редактирования) изображений предназначен компонент TDBImage.

Отдельную группу составляют компоненты синхронного просмотра данных. Они обеспечивают показ значений поля из одной таблицы в соответствии со значениями поля из другой таблицы.

Наконец, данные можно представить в виде графика. Для этого предназначен компонент TDBChart.

Как видите, набор компонентов отображения данных весьма разнообразен и позволяет решать задачи по созданию любых интерфейсов для приложений баз данных.

Ввиду общности решаемых задач, компоненты отображения данных имеют несколько важных общих свойств, которые представлены в табл. 5.1 и в дальнейшем изложении опущены.

Таблица 5.1. Общие свойства компонентов отображения данных

| Объявление | Описание |
|---|---|
| property DataField: string; | Поле связанного с компонентом набора данных |
| property DataSource: TDataSource; | Связываемый с компонентом компонент TDataSource |
| property Field: Tfield; | Обеспечивает доступ к классу TField, который соответствует полю набора данных, заданному свойством DataField |
| property Readonly: Boolean; | Управляет работой режима "только для чтения" |

Табличное представление данных

**Компонент TDBGrid**

Этот компонент инкапсулирует двумерную таблицу, в которой строки представляют собой записи, а столбцы — поля набора данных.

Компонент TDBGrid является потомком классов TDBCustomGrid И TCustomGrid.

От класса TCustomGrid наследуются все функции отображения и управления работой двумерной структуры данных. Класс TDBCustomGrid обеспечивает визуализацию и редактирование полей из набора данных, причем TDBGrid только публикует свойства и методы класса TDBCustomGrid, не добавляя собственных.

В компоненте TDBGrid можно отображать произвольное подмножество полей используемого набора данных, но число записей ограничить нельзя — в компоненте всегда присутствуют все записи связанного набора данных. Требуемый набор полей можно составить при помощи специального Редактора столбцов, который открывается при двойном щелчке на компоненте, перенесенном на форму, или кнопкой свойства columns в Инспекторе объектов.

Новая колонка добавляется при помощи кнопки **Add New**, после этого ее название появляется в списке колонок (рис. 5.2). Для выбранной в списке колонки доступные для редактирования свойства появляются в Инспекторе объектов. Колонки в списке можно редактировать, удалять, менять местами.

При помощи кнопки **Add All Fields** в сетку можно добавить все поля набора данных.

Каждая колонка компонента TDBGrid описывается специальным классом TColumn, а совокупность колонок доступна через свойство columns компонента, оно имеет тип TDBGridColumns и представляет собой индексированный список объектов колонок. Поле набора данных связывается с конкретной колонкой при помощи свойства FieldName класса TColumn. При этом в колонку автоматически переносятся все необходимые параметры поля, в частности заголовок поля, настройки шрифтов, ширина поля. После ручного изменения параметров первоначальные значения восстанавливаются методами соответствующих объектов Icolumn.

При помощи метода DefaultDrawColumnCell и метода- обработчика OnDraw-CoiumnCell можно управлять процессом отображения данных в ячейках.

Метод DefaultorawDataCelll предназначен только для обеспечения обратной совместимости по коду с более ранними версиями.

Настройка параметров компонента TDBGrid, от которых зависит его внешний вид и некоторые функции, осуществляется при помощи свойства options (табл. 5.2). Текущая позиция в двумерной структуре данных может быть определена свойствами SelectedField, SelectedRows, Selectedlndex.

При необходимости разработчик может использовать разнообразные методы-обработчики событий. Среди них есть как стандартные методы, присущие всем элементам управления, так и специфические.

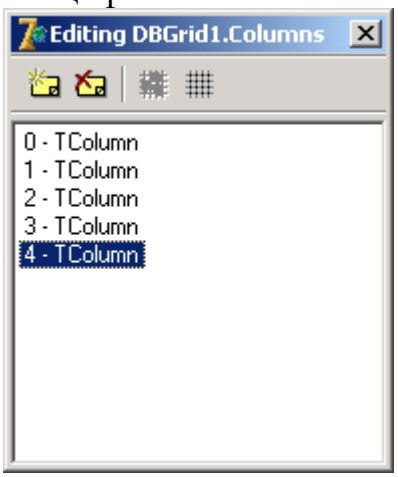


Рисунок 5.2. Редактор колонок компонента TDBGrid

Например, при помощи метода-обработчика OnEditButtonClick можно предусмотреть вызов специализированной формы при щелчке на кнопке в ячейке:

```
procedure TForml.DBGridlEditButtonClick(Sender: TObject);
begin
if DBGridl.Selectedlndex = 2 then SomeForm.ShowModal;
end;
```

**Примечание** *Объект колонки TColumn имеет свойство ButtonStyle. Если ему присвоить значение cbsEllipsis, то при активизации ячейки этой колонки в правой части ячейки появляется кнопка.*

Таблица 5.2. Свойства и методы компонента WBGrid

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property Columns: TDBGridColumns; | Pb | Содержит коллекцию объектов TColumn, описывающих колонки компонента |
| property DefaultDrawing: Boolean; | Pb | Определяет способ визуализации данных в сетке. При значении True данные отображаются автоматически. При значении False используется метод-обработчик OnDrawColumnCell |

| | | |
|---|---|---|
| property FieldCount: Integer; | Ro | Возвращает число видимых колонок сетки |
| property Fields [Index: Integer] : TField; | Ro | Массив объектов полей набора данных, отображаемых в компоненте |
| TDBGridOption = (dgEditing, dgAlwaysShowEditor, dgTitles, dglndicator, dgColumnResize, dgColLines, dgRowLines, dgTabs, dgRowSelect, dgAlwaysShowSelection, dgConfirmDelete, dgCancelOnExit, dgMultiSelect) ; TDBGridOptions = set of TDBGridOption; | Pb | Определяет особенности визуализации и поведения компонента:<br>• dgEditing — данные можно редактировать;<br>• dgAlwaysShowEditor — данные в сетке всегда в режиме редактирования;<br>• dgTitles — видны заголовки колонок;<br>• dglndicator — в начале строки виден номер текущей колонки;<br>• dgColumnResize — колонки можно перемещать и менять их ширину;<br>• dgColLines — видны линии между колонками;<br>• dgRowLines — видны линии между строками;<br>• dgTabs — для перемещения по строкам можно использовать клавиши <Tab> и <Shift>+<Tat»;<br>• dgRowSelect — можно выделять целые строки, при этом игнорируются установки dgEditing И dgAlwaysShowEditor;<br>• dgAlwaysShowSelection — выделение текущей ячейки сохраняется, даже если сетка не активна;<br>• dgConfirmDelete — при удалении строк появляется запрос о подтверждении операции;<br>• dgCancelOnExit — созданные пустые строки при уходе из сетки не сохраняются;<br>• dgMultiSelect — можно выделять несколько строк одновременно |
| property SelectedField: TField; | Pu | Содержит объект текущего поля |
| property Selectedlndex: Integer; | Pu | Содержит номер текущей колонки в массиве свойства Columns |
| property SelectedRows: TBookmarkList; | Ro | Набор закладок на записи набора данных, соответствующих выделенным строкам сетки |

| | | |
|---|---|---|
| property TitleFont: TFont; | Pb | Шрифт заголовков колонок |
| property EditorMode: Boolean; | Pu | Показывает, можно ли редактировать текущую ячейку |
| property FixedColor: TColor; | Pb | Цвет фона неподвижных ячеек сетки |
| **Методы** | | |
| procedure DefaultDraw-ColumnCell (const Rect: TRect; DataCol: Integer; Column: TColumn; State: TGridDrawState) ; | Pu | Перерисовывает текст в ячейке колонки с номером DataCol. Ячейка задается прямоугольником Rect на канве сетки. Параметр state определяет состояние ячейки после перерисовки. Параметр Column содержит экземпляр класса колонки, которой принадлежит ячейка |
| procedure DefaultDrawDataCell (const Rect: TRect; Field: TField; State: TGridDrawState); | Pu | Перерисовывает текст в ячейке колонки, определяемой параметром Field, содержащим связанный с колонкой объект поля. Ячейка задается прямоугольником Rect на канве сетки. Параметр State определяет состояние ячейки после перерисовки |
| procedure Def aultHandler (var Msg); override; | Pu | Вызывает всплывающее меню для колонки, которой соответствуют текущие координаты мыши. Компонент должен обрабатывать сообщение WM RBUTTONUP |
| function ExecuteAction (Action: TBasicAction): Boolean; over-ride; | Pu | Выполняет действие, заданное параметром Action, по отношению к данному компоненту |
| procedure ShowPopupEditor (Column: I TColumn; X: Integer = Low (Integer); Y: Integer = Low (Integer) ); dynamic; | Pu | Открывает набор данных, связанный с передаваемой параметром Column колонкой в новом окне. Работает только для типов данных абстрактный и набор данных. Параметры X и Y определяют положение нового окна |
| function \| ValidFieldlndex (Fieldlndex: Integer) : Boolean; | Pu | Возвращает значение True, если колонка с номером Fieldlndex связана с полем набора данных |
| type TGridCoord = record X: Longint;Y: Longint; end; function MouseCoord(X, Y: In-teger): TGridCoord; ; | Pu | Возвращает номера строки и столбца, соответствующие ячейке, которой принадлежат экранные координаты X и Y |
| **Методы-обработчики событий** | | |

| | | |
|---|---|---|
| type TDBGridClickEvent = procedure (Column: TColumn) 1 of object;<br>property OnCellClick: TDBGridClickEvent; | Pb | Вызывается при щелчке мышью на ячейке. Параметр Column содержит колонку, которой принадлежит ячейка |
| property OnColEnter: TNotifyEvent; | Pb | Вызывается при переносе фокуса на новую колонку сетки |
| property OnColExit: TNotifyEvent; | Pb | Вызывается перед переносом фокуса из текущей колонки |
| type TMovedEvent = procedure (Sender: TObject; Froinlndex, Tolndex: Longint) of object;<br>property OnColumnMoved: TMovedEvent ; | Pb | Вызывается при переносе колонки в сетке на новое место при помощи мыши. Параметр Fromindex возвращает номер старого положения колонки. Параметр Tolndex возвращает номер нового положения колонки |
| type TDrawColumnCellEvent = procedure (Sender: TObject; const Rect : TRect; DataCol : State: TGridDrawState) of object;<br>property OnDrawColurnnCell : TDrawColumnCellEvent; | Pb | Вызывается при перерисовке ячейки.<br>Параметр Rect определяет ячейку по координатам прямоугольника на канве.<br>Параметр DataCol возвращает номер колонки в сетке.<br>Параметр Column содержит объект колонки.<br>Параметр State возвращает состояние колонки |
| type TDrawDataCellEvent = procedure (Sender: TObject; const Rect: TRect; Field: TField; State: TGridDrawState) of object;<br>property OnDrawDataCell: TDrawDataCellEvent; | Pb | Вызывается при перерисовке ячейки перед обработчиком OnDrawCoiumnCell, если свойство Columns. State =csDefault.<br>Этот метод лучше не применять, т. к. он используется только для обеспечения обратной совместимости с ранними версиями |
| property OnEditButtonClick: TNotifyEvent; | Pb | Вызывается при щелчке мышью на кнопке в ячейке |
| type TDBGridClickEvent = procedure (Column: TColumn) of object;<br>property OnTitleClick: TDBGridClickEvent; | Pb | Вызывается при щелчке мышью на заголовке колонки. Колонка определяется параметром Column |

В работе компонента TDBGrid важную роль играет класс TColumn, который инкапсулирует свойства колонки или столбца сетки (табл. 5.3). Его основным назначением является правильное отображение данных из поля набора данных, связанного с этой колонкой. Поэтому объект колонки обладает свойствами и методами, которые позволяют произвольным образом задавать параметры отображения данных (цвет, шрифт, ширину и т. д.). Первоначальные значения берутся из связанных с колонками полей. Измененные свойства можно восстановить при помощи группы специальных методов (DefauitColor, DefaultFont и др.).

Свойство Assignedvalues позволяет в любой момент определить, какие первоначальные настройки были изменены.

За отображение заголовка колонки отвечает свойство Title, представляющее собой ссылку на экземпляр объекта TColumnTitie. Здесь можно задать текст заголовка, параметры шрифта текста заголовка и цвет фона заголовка. По умолчанию текст заголовка берется из свойства DispiayLabel объекта TField (СМ. гл. 13).

Каждой колонке можно придать список, который разворачивается при щелчке на кнопке в активной ячейке колонки. Выбранное в списке значение автоматически заносится в ячейку. Для реализации этой возможности применяется свойство pickList типа TStrings. Достаточно лишь заполнить список значениями во время разработки или выполнения (рис. 5.3).

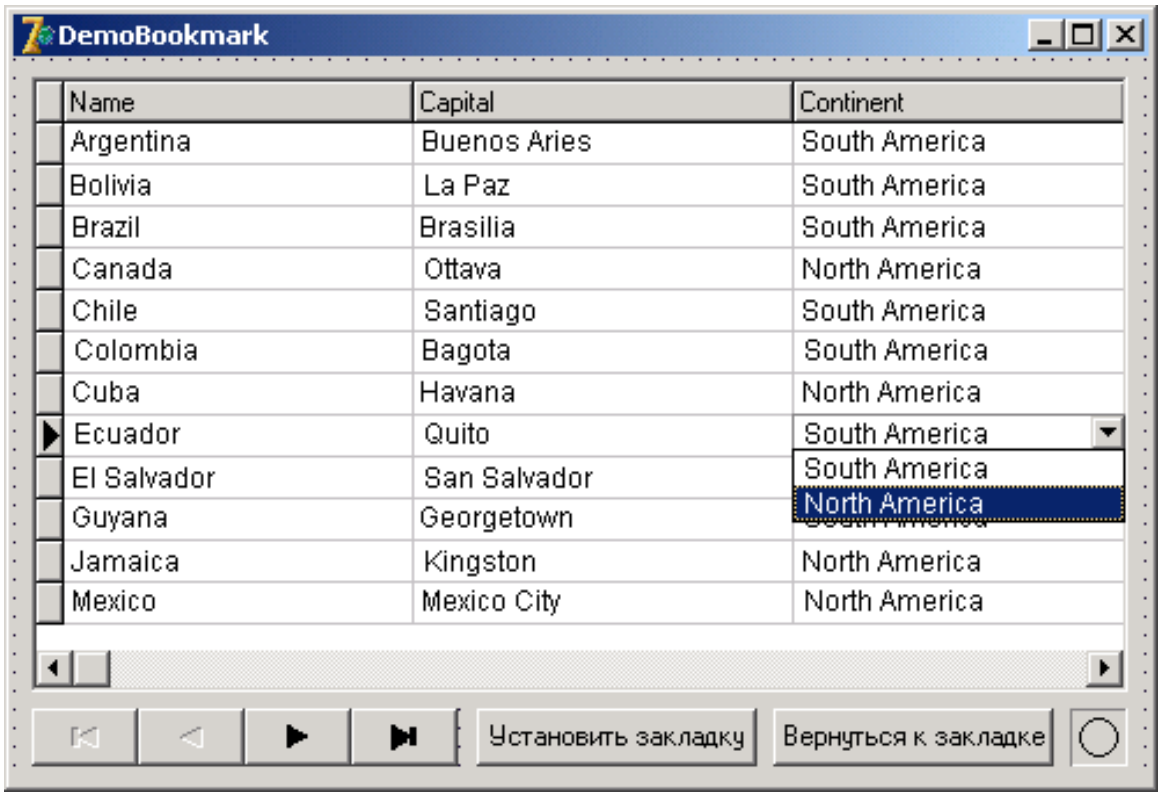


Рисунок 5.3. Список колонки в компоненте TDBGrid

Таблица 5.3. Свойства и методы класса TColumn

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property Alignment: TAlignment; | Pb | Определяет выравнивание данных в колонке |
| type TColumnValue = (cvColor, cvWidth, cvFont, cvAlignment, cvReadOnly, cvTitleColor, cvTitleCaption, cv'i'itleAlignment, cvTitleFont, cvImeMode, cvImeName) ; TColuinnValues = set of TColumnValue; property AssignedValues: TColuranValues ; | Ro | Возвращает набор атрибутов колонки, которые были изменены по сравнению с первоначальными |
| type TColumnButtonStyle = (cbsAuto, cbsEllipsis, cbsNone) ; property ButtonStyle: TColumnButtonStyle; | Pb | Задает способ редактирования данных в колонке:<br>• cbsAuto — кнопка в редактируемой ячейке появляется, если связанное поле является полем синхронного просмотра;<br>• cbsEllipsis — кнопка в редактируемой ячейке появляется всегда, щелчок на кнопке вызывает обработчик OnEditButtonClick;<br>• cbsNone — при редактировании ячейки кнопка не появляется |
| property Color: TColor; | Pb | Цвет фона колонки |
| property DisplayName: string; | Pu | Название колонки в списке Редактора столбцов |
| property DropDownRows: Cardinal; | Pb | Определяет число строк разворачивающегося списка ячейки |
| property Expandable: Boolean; | Pu | В значении True разрешает показ связанных с полем колонки дочерних полей абстрактного, ссылочного типов и массивов |
| property Expanded: Boolean; | Pb | При значении True каждое дочернее поле отображается в новой колонке. При значении False дочерние поля отображаются через точку с запятой и не доступны для редактирования |

| | | |
|---|---|---|
| property FieldName: string; | Pb | Название поля, связанного с колонкой |
| property Font: TFont; | Pb | Шрифт данных в колонке |
| property Grid: TCustomDBGrid; | Ro | Определяет сетку, содержащую эту колонку |
| property ParentColumn: TColumn; | Ro | Определяет колонку-владельца текущей колонки. Используется для дочерних полей |
| property PickList: TStrings; | Pb | Содержит разворачивающийся список, используемый при редактировании данных |
| property PopupMenu: TPopupMenu; | Pb | Связывает с колонкой всплывающее меню |
| property Showing: Boolean; | Ro | Возвращает значение True, если колонка видима |
| property Title: TColumnTitle; | Pb | Задает текст заголовка и его параметры |
| property Visible: Boolean; | Pb | Задает видимость колонки |
| property Width: Integer; | pb | Задает ширину колонки в пикселах |
| **Методы** | | |
| procedure Assign (Source: TPersistent); override; | Pu | Копирует колонку Source в текущую колонку |
| function Def aultAlignment: TAlignment; | Pu | Возвращает первоначальное значение выравнивания колонки |
| function DefaultColor: TColor; | Pu | Возвращает первоначальный фоновый цвет колонки |
| function DefaultFont: TFont; | Pu | Возвращает первоначальный шрифт данных в колонке |
| type TIraeMode = (imDisable, iraClose, imOpen, im-DontCare, imSAlpha, imAl-pha, imHira, imSKata, irnKa-ta, imChinese, imSHanguel, imHanguel); function Default-lmeMode: TImeMode; | Pu | Возвращает первоначальный способ ввода символов |
| type TImeName = type string; function DefaultlmeName: TImeName; | Pu | Возвращает первоначальное имя редактора способа ввода символов |
| function DefaultReadOnly: Boolean; | Pa | Возвращает первоначальный режим редактирования данных |

| | | |
|---|---|---|
| function DefaultWidth: Integer; | Pu | Возвращает первоначальную ширину колонки в пикселах |
| function Depth: Integer; | Pu | Возвращает число непосредственных предков колонки |
| procedure RestoreDefaults; virtual; | Pu | Восстанавливает первоначальные настройки колонки |

При работе с компонентом TDBGrid все операции с отдельными колонками осуществляются при помощи экземпляра класса TDBGridColumns, который инкапсулирует список объектов колонок (свойство Columns компонента TDBGrid). Доступ к колонкам осуществляется при помощи свойства items. Нумерация колонок начинается с нуля.

При помощи свойств и методов класса TDBGridColumns можно изменять настройки полей компонента TDBGrid во время выполнения (табл. 5.4).

Свойство state определяет способ создания колонок. Его значение устанавливается автоматически. При создании колонок для всех полей сразу (кнопка **Add All Fields** Редактора столбцов) устанавливается значение csDefault.

При любом ручном изменении свойств устанавливается значение csCustomized. При программном изменении значения свойства во время выполнения все существующие колонки удаляются.

Все данные из существующих колонок можно сохранить в файле или потоке при помощи методов SaveToFile и saveToStream, а затем загрузить их обратно методами LoadFromFile И LoadFromStream.

Таблица 5.4. Свойства и методы класса TDBGridColumns

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property Grid: TCustomDBGrid; | Ro | Возвращает ссылку на сетку, владеющую данным объектом |
| property Items [Index: Integer] : TColumn default; | Pu | Индексный список объектов колонок сетки: |
| type TDBGridColumnsState = (csDefault, csCustomized) ;<br>property State: TDBGridCol-umnsState; | Pu | Определяет способ создания колонок сетки:<br>• csDefault — колонки создаются динамически с параметрами, соответствующими связанным полям;<br>• csCustomized — параметры колонок определены разработчиком и могут отличаться от параметров полей |
| property Count: Integer; | Pu | Возвращает общее число колонок |
| **Методы** | | |
| function Add: TColumn; | Pu | Добавляет новый объект TColumn |

| | | |
|---|---|---|
| procedure LoadFromFile (const Filename: string); | Pu | Загружает данные в объект из файла FileName |
| procedure LoadFromStream(S: TStream) ; | Pu | Загружает данные в объект из потока s |
| procedure RebuildColumns; | Pu | Удаляет существующие колонки и создает новые, основываясь на параметрах полей набора данных |
| procedure RestoreDefaults; | Pu | Восстанавливает первоначальные настройки колонок |
| procedure SaveToFiie (const Filename: string); | Pu | Сохраняет данные из колонок в файле FileName |
| procedure SaveToStream(S: TStream) ; | Pu | Сохраняет данные из колонок в потоке s |

**Компонент TDBCtrlGrid**

Компонент TDBCtrlGrid внешне напоминает компонент TDBGrid, но никак не связан с классом TCustomDBGrid, а наследуется напрямую от класса TWinControl.

Этот компонент позволяет отображать данные в строках в произвольной форме. Компонент представляет собой набор панелей, каждая из которых служит платформой для размещения данных отдельной записи набора данных. На панели могут размещаться любые компоненты отображения данных, предназначенные для работы с отдельным полем. С каждым таким компонентом можно связать нужное поле набора данных. При открытии набора данных в компоненте TDBCtrlGrid на каждой новой панели создается набор компонентов отображения данных, аналогичный тому, который был создан на одной панели во время разработки.

На панель можно переносить только те компоненты отображения данных, которые показывают значение одного поля для единственной записи набора данных**.** Нельзя использовать компоненты TDBGrid, TDBCtrlGrid, TDBRichEdit, TDBListBox, TDBRadioGroup, TDBLookupListBox.

После того, как для компонента TDBCtrlGrid задано значение свойства DataSource, все переносимые на панель компоненты отображения данных автоматически связываются с указанным компонентом TdataSource (табл. 5.5). Самостоятельное задание свойства DataSource для дочерних компонентов отображения данных не допускается. В них требуется определить только поля.

Компонент может отображать панели в одну или несколько колонок. Для задания числа колонок панелей используется свойство colcount. Число видимых строк панелей определяется свойством RowCount. Вертикальное или горизонтальное размещение колонок панелей зависит от значения свойства Orientation.

При использовании нескольких колонок панелей курсор перемещается по колонке сверху вниз с последующим переходом на следующую колонку. Направление движения курсора не зависит от значения свойства Orientation.

Размеры одной панели определяются свойствами panelHeight и Panelwidth. Они взаимосвязаны с размерами самого компонента. При изменении значений свойств PanelHeight и Panelwidth размеры компонента изменяются таким образом, чтобы в нем помещалось указанное в свойствах colcount и RowCount число панелей и наоборот.

Не рекомендуется размещать на панели компоненты TDBMemo и TDBimage, т. к. это может привести к значительному снижению производительности.

Таблица 5.5. Свойства и методы компонента TDBCtrlGrid

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property AllowDelete: Boolean; | Pb | Разрешает или запрещает удаление текущей записи |
| property Allowlnsert: Boolean; | Pb | Разрешает или запрещает вставку новой записи |
| property Canvas: TCanvas; | Ro | Канва компонента |
| property ColCount: Integer; | Pb | Определяет число колонок с панелями |
| property EditMode: Boolean; | Pu | Разрешает или запрещает редактирование данных |
| type TDBCtrlGridOrientation = (goVertical, goHorizontal);<br>property Orientation: TDBCtrlGridOrientation; | Pb | Определяет порядок следования записей — по горизонтали или по вертикали |
| type TDBCtrlGridBorder = (gbNone, gbRaised) ;<br>property PanelBorder: TDBCtrlGrid-Border; | Pb | Определяет способ отображения границы панели |
| property PanelCount: Integer; | Ro | Содержит число видимых одновременно панелей |
| property PanelHeight: Integer; | Pb | Определяет высоту панелей в пикселах |
| property Panellndex: Integer; | Pu | Определяет индекс панели текущей записи |
| property PanelWidth: Integer; | Pb | Определяет ширину панелей в пикселах |

| | | |
|---|---|---|
| property RowCount: Integer; | Pb | Определяет число строк видимых панелей |
| property SelectedColor: TColor; | Pb | Определяет фоновый цвет панели текущей записи |
| property ShowFocus : Boolean; | Pb | Разрешает или запрещает выделение вокруг панели текущей записи |
| **Методы** | | |
| type TDBCtrlGridKey = (gkNull, gkEditMode, gkPriorTab, gkNextTab, gkLeft, gkRight, gkUp, gkDown, gkScrollUp, gkScrollDown, gkPageUp, gkPageDown, gkHome, gkEnd, gklnsert, gkAppend, gkDelete, gkCancel) ;<br>procedure DoKey(Key: TDBCtrlGridKey) ; | | Выполняет операцию, заданную при помощи параметра Key.<br>Доступны операции навигации по записям, перевода в режим редактирования, вставки, удаления записей, отмены изменений |
| procedure KeyDown (var Key: Word; Shift: TShiftState) ; override; | | Используется при нажатии клавиши для трансляции кодов клавиш |
| **Методы-обработчики событий** | | |
| type TPaintPanelEvent = procedure (DBCtrlGrid: TDBCtrlGrid; Index: Integer) of object;<br>property OnPaintPanel : TPaintPanelEvent; | | Вызывается при перерисовке панели. Параметр Index соответствует индексу панели |

Навигация по набору данных

Перемещение или навигация по записям набора данных может осуществляться несколькими путями. Например, в компонентах TDBGrid и TDBCtrlGrid, которые отображают сразу несколько записей набора данных, можно использовать клавиши вертикального перемещения курсора или вертикальную полосу прокрутки.

Но что делать, если на форме находятся только компоненты, отображающие одно поле только текущей записи набора данных (TDBEdit, TDBComboBox и т. д.)? Очевидно, что в этом случае на форме должны быть расположены дополнительные элементы управления, отвечающие за перемещение по записям.

Аналогично, ни один компонент отображения данных не имеет встроенных средств для создания и удаления записей целиком.

Для решения указанных задач и предназначен компонент TDBNavigator, который представляет собой совокупность управляющих кнопок, выполняет операции навигации по набору данных и модификации записей целиком.

Компонент TDBNavigator при помощи свойства DataSource связывается с компонентом TDataSource и через него с набором данных. Такая схема позволяет обеспечить изменение текущих значений полей сразу во всех связанных с TDataSource компонентах отображения данных. Таким образом, TDBNavigator только дает команду на выполнение перемещения по набору данных или другой управляющей операции, а всю реальную работу выполняют компонент набора данных и компонент TDataSource. Компонентам отображения данных остается только принять новые данные от своих полей.

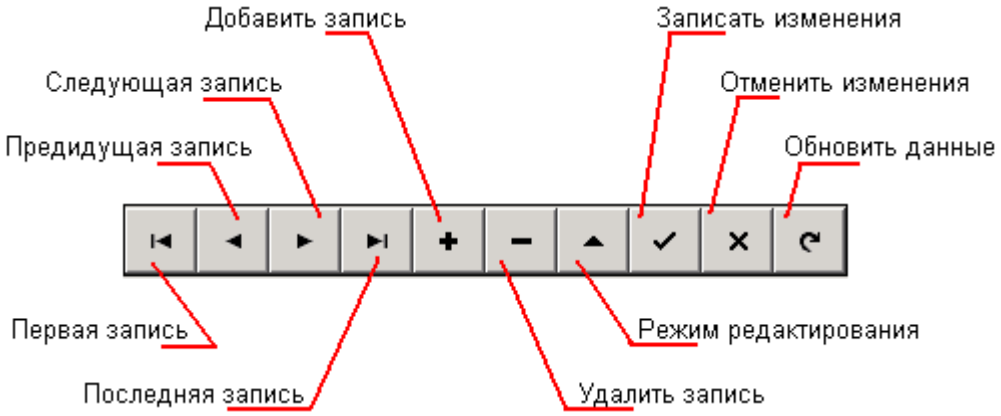


Рисунок 5.4. Назначение кнопок компонента TDBNavigator

Компонент TDBNavigator содержит набор кнопок, каждая из которых отвечает за выполнение одной операции над набором данных. Всего имеется 10 кнопок, разработчик может оставить в наборе любое количество кнопок в любом сочетании. Видимостью кнопок управляет свойство visibleButtons:

```
type
TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nblnsert, nbDelete,
nbEdit, nbPost, nbCancel, nbRefresh);
TButtonSet = set of TNavigateBtn;
property VisibieButtons: TButtonSet;
```

Каждый элемент типа TNavigateBtn представляет одну кнопку, их назначение описывается ниже:

nbFirst — перемещение на первую запись набора данных;

nbPrior — перемещение на предыдущую запись набора данных;

nbNext — перемещение на следующую запись набора данных;

nbLast — перемещение на последнюю запись набора данных;

nblnsert — вставка новой записи в текущей позиции набора данных;

nbDelete — удаление текущей записи, курсор перемешается на следующую запись;

nbEdit — набор данных переводится в режим редактирования;

nbPost — в базу данных переносятся все изменения в текущей записи;

nbcancel — все изменения в текущей записи отменяются;

nbRefresh — восстанавливаются первоначальные значения текущей записи, сделанные после последнего переноса изменений в базу данных.

Самой критичной к возможной потере данных вследствие ошибки является операция удаления записи, поэтому при помощи свойства confirmDelete можно включить механизм контроля удаления. При каждом удалении записи нужно будет дать подтверждение выполняемой операции.

Нажатие любой кнопки можно эмулировать программно при помощи метода BtnClick.

В случае необходимости выполнения дополнительных действий при щелчке на любой кнопке можно воспользоваться обработчиками событий BeforeAction и Onciick, в которых параметр Button определяет нажатую кнопку. Свойства и методы компонента TDBNavigator представлены в табл. 5.6.

Таблица 15.6. Свойства и методы компонента TDBNavigator

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property ConfirmDelete: Boolean; | Pb | Включает или отключает подтверждение удаления записи |
| property Hints: TStrings; | Pb | Содержит список подсказок для каждой кнопки |
| property Flat: Boolean; | Pb | Определяет внешний вид кнопок компонента |
| type TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nblnsert, nbDeiete, nbEdit, nbPost, nbCancel, nbRefresh); TButtonSet = set of TNavi-gateBtn ; property VisibleButtons: TBut-tonSet; | Pb | Список видимых кнопок |
| **Методы** | | |
| procedure BtnClick (Index: TNavigateBtn) ; | Pu | Эмулирует щелчок на кнопке index |
| procedure SetBounds (ALeft, ATop, AWidth, AHeight: Inte-ger) ; | Pu | Задает положение (параметры ALeft, ATop) и размер компонента (параметры AWidth, AHeight) |
| **Методы-обработчики событий** | | |
| ENavClick = procedure (Sender: TObject; Button: TNavigateBtn) of object; Iproperty BeforeAc-tion: ENavClick; | Pb | Выполняется при щелчке на кнопке ' Button перед выполнением операции, связанной с кнопкой |

| | | |
|---|---|---|
| ENavClick = procedure (Sender: TObject; Button: TNavigateBtn) of object;<br>property OnClick: ENavClick; | Pb | Выполняется при щелчке на кнопке Button после выполнения операции, связанной с кнопкой |

*Представление отдельных полей*

Большинство компонентов отображения данных предназначено для представления данных из отдельных полей. Для этого все они имеют свойство DataField, которое указывает на требуемое поле набора данных.

В зависимости от типа данных поля могут использовать различные компоненты. Для большинства стандартных полей используются компоненты TDBText, TDBEdit, TDBComboBox, TDBListBox.

Данные в формате Memo отображаются компонентами TDBMemo и TDBRichEdit. Для показа изображений предназначен компонент TDBImage.

**Компонент TDBText**

Этот компонент представляет собой статический текст, который отображает текущее значение некоторого поля связанного набора данных. При этом данные можно просматривать в режиме "только для чтения".

Непосредственным предком компонента является класс TCustomLabel, поэтому он очень похож на компонент TLabel.

При использовании компонента следует обратить внимание на возможную длину отображаемых данных. Для предотвращения обрезания текста можно использовать свойства AutoSize И Wordwrap.

**Компонент TDBEdit**

Компонент представляет собой стандартный однострочный текстовый редактор, в котором отображаются и изменяются данные из поля связанного набора данных.

Прямой предок компонента — класс TCustomMaskEdit, который также является прямым предком компонента TEdit.

Компонент может осуществлять проверку редактируемых данных по заданной для поля маске. Непосредственно для редактора задать маску нельзя, т. к. содержащее маску свойство EditMask в классе TCustomMaskEdit является защищенным, а в TDBEdit не перекрыто. Тем не менее механизм контроля полностью унаследован. Саму же маску можно задать в связанном с редактором поле. Объект TField имеет собственное свойство EditMask, которое и используется при проверке данных в редакторе.

Проверка редактируемого текста на соответствие маске осуществляется методом validateEdit после каждого введенного или измененного символа. В случае ошибки генерируется исключение validateError и курсор устанавливается на первый ошибочный символ.

В компоненте можно использовать буфер обмена. Это делается средствами операционной системы пользователем или программно при помощи методов CopyToClipboard, CutToClipboard, PasteFromCiipboard.

**Компонент TDBCheckBox**

Компонент представляет собой почти полный аналог обычного флажка (компонент TCheckBox) и предназначен для отображения и редактирования любых данных, которые могут иметь только два значения. Это может быть логический тип данных или любые строковые значения, но поле может принимать значения только из двух строк.

Предопределенные значения задаются свойствами valuechecked и ValueUnchecked. По умолчанию они имеют значения True И False. Этим свойствам можно также присваивать любые строковые значения, причем одному свойству можно назначить несколько возможных значений, разделенных точкой с запятой.

Включение флажка происходит, если значение поля набора данных совпадает со значением свойства valuechecked (единственным или любым из списка). Если же флажок включил пользователь, то значение поля данных приравнивается к единственному или первому в списке значению свойства ValueChecked.

Аналогичные действия происходят и со свойством ValueUnchecked.

**Компонент TDBRadioGroup**

Компонент представляет собой стандартную группу переключателей, состояние которых зависит от значений поля связанного набора данных. В поле можно передавать фиксированные значения, связанные с отдельными переключателями в группе.

Если текущее значение связанного поля соответствует значению какого-либо переключателя, то он включается. Если пользователь включает другой переключатель, то связанное с переключателем значение заносится в поле. Возможные значения, на которые должны реагировать переключатели в группе, заносятся в свойство Values при помощи специального редактора в Инспекторе объектов или программно посредством методов класса Tstrings. Каждому элементу свойства values соответствует один переключатель (порядок следования сохраняется).

Свойство items содержит список поясняющих надписей для переключателей группы. Если для какого-либо переключателя нет заданного значения, но есть поясняющий текст, то такой переключатель включается при совпадении значения связанного поля с поясняющим текстом.

Текущее значение связанного поля содержится в поле value.

**Компонент TDBListBox**

Компонент отображает текущее значение связанного с ним поля набора данных и позволяет изменить его на любое фиксированное из списка. Функционально компонент ничем не отличается от компонента TListBox. Значение поля должно совпадать с одним из элементов списка. Специальных методов компонент не содержит.

**Компонент TDBComboBox**

Компонент отображает текущее значение связанного с ним поля набора данных в строке редактирования, при этом значение поля должно совпадать с одним из элементов разворачивающегося списка. Текущее значение можно изменить на любое фиксированное из списка компонента. Функционально

компонент ничем не отличается от компонента TDBCombovox, представляющего собой комбинированный список.

Компонент может работать в пяти различных стилях, которые определяются свойством Style.

Специальных методов компонент не содержит.

**Компонент TDBMemo**

Компонент представляет собой обычное поле редактирования, к которому подключается поле с типом данных Memo или BLOB. Основное его преимущество — возможность одновременного просмотра и редактирования нескольких строк переменной длины. Компонент может отображать только строки, которые целиком видны по высоте.

В компоненте можно использовать буфер обмена при помощи стандартных средств операционной системы или унаследованными от предка TCustomMemo методами CopyToClipBoard, CutToClipBoard, PasteFromClipBoard.

Для ускорения навигации по набору данных при отображении полей типа BLOB можно использовать свойство AutoDisplay. При значении True любое новое значение поля автоматически отображается в компоненте. При значении False новое значение появляется только после двойного щелчка на компоненте или после нажатия клавиши <Enter> при активном компоненте.

Метод LoadMemo используется автоматически при загрузке значения поля, если свойство AutoDispiay = False.

Поведением компонента при работе со слишком длинными строками можно управлять при помощи свойства wordwrap. При значении True слишком длинная строка сдвигается влево при перемещении текстового курсора за правую границу компонента. При значении False остаток длинной строки переносится на новую строку, при этом реально новая строка в данных не создается.

**Компонент TDBImage**

Компонент предназначен для просмотра изображений, хранящихся в базах данных в графическом формате.

Редактировать изображения можно только в каком-либо графическом редакторе, перенося исходное и измененное изображение при помощи буфера обмена. Это делается средствами операционной системы пользователем или программно при помощи методов CopyToClipboard, CutToClipboard, PasteFromClipboard.

Визуализация изображения осуществляется при помощи свойства Picture, которое представляет собой экземпляр класса TPicture.

Также можно полностью заменить существующее изображение или сохранить новое в новой записи набора данных. Для этого используются методы свойства Picture.

Свойство AutoDisplay позволяет управлять процессом загрузки новых изображений из набора данных в компонент. При значении True любое новое значение поля автоматически отображается в компоненте. При значении False

новое значение появляется только после двойного щелчка на компоненте или после нажатия клавиши <Enter> при активном компоненте.

Для ускорения просмотра изображений можно применять свойство QuickDraw, которое задает используемую изображением палитру. При значении True применяется стандартная системная палитра. В результате уменьшается время загрузки изображения, но может ухудшиться и качество изображения, в некоторых случаях до полного искажения. При значении False используется собственная палитра изображения и процесс загрузки замедляется.

**Компонент TDBRichEdit**

Компонент предоставляет возможности полноценного текстового редактора для просмотра и изменения текстовых данных, хранящихся в связанном поле набора данных. Поле должно содержать информацию о форматировании текста.

Внешне компонент ничем не отличается от поля редактирования, поэтому о реализации доступа к гораздо более богатым возможностям редактора через интерфейс пользователя должен позаботиться разработчик. Для этого можно использовать дополнительные элементы управления.

**Синхронный просмотр данных**

При разработке приложений для работы с базами данных часто возникает необходимость в связывании двух наборов данных по ключевому полю. Например, в таблице Orders (содержит данные о заказах) демонстрационной базы данных DBDEMOS имеется поле custNo, которое содержит идентификационный номер покупателя. Под этим же номером в таблице Customers хранится информация о покупателе (адрес, телефон, реквизиты и т. д.). При разработке пользовательского интерфейса приложения баз данных необходимо, чтобы при просмотре перечня заказов в форме приложения отображались не идентификационные номера покупателей, а их параметры.

Таким образом, в наборе данных заказов вместо поля номера покупателя должно появиться поле имени покупателя из таблицы Customers. Механизм связывания полей из различных наборов данных по ключевому полю называется синхронным просмотром. В рассмотренном примере ключевым является поле CustNo из таблицы Customers, а выбор конкретного наименования производится по совпадению значений ключевого поля и заменяемого поля из исходного набора данных — Orders. Причем необходимо, чтобы в таблице Customers поле custNo было уникальным (составляло первичный или вторичный ключ).

Таблицу, в которой расположено поле, значения которого замещаются на синхронные, будем называть исходной таблицей (это таблица Orders).

Таблицу, содержащую ключевое поле и поле данных для синхронного просмотра, будем называть таблицей синхронного просмотра (таблица Customers).

В Delphi механизм синхронного просмотра реализован на уровне отдельных полей и компонентов.

В наборе данных динамически можно создать специальное поле синхронного просмотра, которое будет автоматически замещать одно значение другим в

зависимости от значения ключевого поля. Такое поле можно связать с любым рассмотренным выше компонентом отображения данных.

Помимо простого синхронного просмотра данных может возникнуть задача редактирования данных в аналогичной ситуации. Для этого предназначены специальные компоненты синхронного просмотра данных, которые позволяют, например, выбирать покупателя из списка, а изменится при этом номер покупателя в наборе данных заказов. Использование таких компонентов делает пользовательский интерфейс значительно более удобным и наглядным. В VLC Delphi есть два таких компонента: TDBLookupListBox И TDBLookupComboBox.

**Примечание** *На странице Win 3.1 Палитры компонентов имеются еще два компонента: TDBLokupList и TDBLookupCombo. Они обладают тем же набором функций, используются для обеспечения совместимости с приложениями, созданными в среде разработки Delphi 1, и поэтому здесь не рассматриваются.*

**Механизм синхронного просмотра**

Непосредственным предком компонентов синхронного просмотра данных является класс TDBLookupControl, который инкапсулирует список значений для просмотра и сам механизм синхронного просмотра.

Как и в любом другом компоненте отображения данных, в компонентах синхронного просмотра должны присутствовать средства связывания с требуемым полем некоторого набора данных (табл. 5.7). Это уже известные свойства: Datasource - применяется для задания набора данных через компонент TDataSource и DataField - для определения требуемого поля набора данных. Для синхронного просмотра следует выбирать такое поле, значения которого не дают пользователю полной информации об объекте и совпадают с ключевым полем в таблице синхронного просмотра. Название этого поля может не совпадать с названием ключевого поля, но типы данных должны быть одинаковыми.

**Примечание**

*При проектировании баз данных желательно, чтобы такие поля все же носили одинаковые названия.*

Теперь необходимо задать таблицу синхронного просмотра, ключевое поле и поле синхронного просмотра.

Набор данных, содержащий указанные поля, определяется через соответствующий компонент TDataSource в свойстве ListSource.

Ключевое поле задается свойством KeyField. Во время работы компонента в свойстве KeyValue содержится текущее значение, которое связывает между собой два набора данных.

Поле синхронного просмотра определяется свойством ListField. Здесь можно задавать сразу несколько полей, которые будут отображаться в компоненте синхронного просмотра. Названия полей разделяются точкой с запятой. Если свойство не определено, то в компоненте будут отображаться значения ключевого поля. Свойство ListFieldindex служит для выбора основного поля из списка. Дело в том, что компоненты синхронного просмотра поддерживают механизм наращиваемого поиска, который позволяет быстро находить нужное значение в больших списках. Свойство ListFieldindex определяет, какое поле используется при наращиваемом поиске. В компоненте TDBiookupComboBox свойство

ListFieldindex также определяет, какое поле будет передано в строку редактирования.

Таблица 5.7. Основные свойства, включающие механизм синхронного просмотра

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| property KeyFieid: string; | Pb | Ключевое поле таблицы синхронного просмотра |
| property KeyValue: Variant; | Pu | Текущее значение ключевого поля |
| property ListFieid: string; | Pb | Поле или список полей синхронного просмотра в таблице синхронного просмотра |
| property ListFieidindex: Integer; | Pb | Номер основного поля синхронного просмотра (используется, когда свойство ListField содержит список полей) |
| property ListSource: TDataSource; | Pb | Указывает на компонент TDataSource, связанный с таблицей синхронного просмотра |
| property NullValueKey: TShortCut; | Pb | Определяет комбинацию клавиш, нажатие которых задает нулевое значение поля |

В качестве примера рассмотрим приложение Demo Lookup (рис. 5.5), в котором с набором данных таблицы Orders из базы данных DBDEMOS связаны компоненты TDBGrid и TDBLookuppComboBox. Во втором компоненте при перемещении по записям набора данных отображается имя покупателя, оформившего текущий заказ.

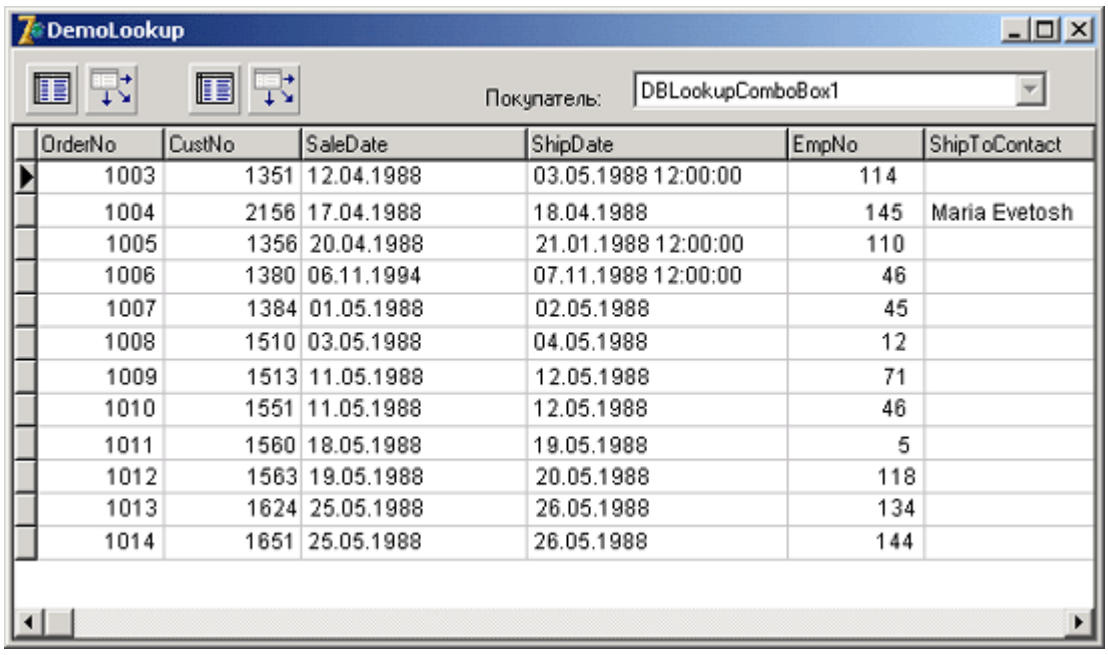

Рисунок 5.5. Главная форма проекта DemoLookup

Ключевые свойства компонента настроены следующим образом.

Свойство Listsource указывает на компонент custsource типа TDataSource, который связан с набором данных синхронного просмотра custTable.

Свойство ListFieid указывает на поле company, все значения которого доступны в списке компонента.

Свойство KeyField указывает на поле custNo, которое имеется в двух таблицах и по которому осуществляется связь.

Рассмотрим основные свойства и методы самих компонентов отображения данных, за исключением тех, которые представлены в табл. 5.7 и полностью идентичны для двух компонентов.

**Компонент TDBLookupListBox**

Компонент представляет собой список значений поля синхронного просмотра для поля, заданного свойством DataField, из набора данных DataSource. Его основное назначение — автоматически устанавливать соответствие между полями двух наборов данных по одинаковому значению заданного поля исходной таблицы и ключевого поля таблицы синхронного просмотра. В списке синхронного просмотра отображаются возможные значения для редактирования поля основной таблицы.

По своим функциональным возможностям компонент совпадает с компонентом TDBListBox.

**Компонент TDBLookupComboBox**

Компонент представляет собой комбинированный список значений поля синхронного просмотра для поля, заданного свойством DataField, из набора данных DataSource. Его основное назначение — автоматически устанавливать соответствие между полями двух наборов данных по одинаковому значению заданного поля исходной таблицы и ключевого поля таблицы синхронного просмотра. В списке синхронного просмотра отображаются возможные значения для редактирования поля основной таблицы.

По своим функциональным возможностям компонент совпадает с компонентом TDBComboBox.

Графическое представление данных

Для представления данных из некоторого набора данных в виде графиков различных видов предназначен компонент TDBChart (табл. 5.8). В нем можно одновременно показывать графики для нескольких полей данных. Графики строятся на основе всех имеющихся в наборе данных значений полей. Функционально компонент ничем не отличается от компонента TChart.

Настройка параметров компонента осуществляется специальным редактором, который можно открыть двойным щелчком на перенесенном на форму компоненте.

Здесь мы не будем подробно останавливаться на богатейших изобразительных возможностях этого компонента, рассмотрим только процесс подключения к нему набора данных и построение графиков.

Основой любого графика в компоненте TDBChart является так называемая серия, свойства которой представлены классом Tchartseries. Для того чтобы построить график значений некоторого поля набора данных, необходимо

выполнить следующие действия, большинство из которых выполняется в специализированном редакторе компонента.

1. Создать новую серию и определить ее тип.
2. Задать для серии набор данных.
3. Связать с осями координат нужные поля набора данных и, в зависимости от типа серии, задать дополнительные параметры.
4. Открыть набор данных.

Редактор имеет две главные страницы — **Chart и Series.** Страница **Chart** содержит многостраничный блокнот и предназначена для настройки параметров самого графика. Страница **Series** также содержит многостраничный блокнот и используется для настройки серий значений данных.

Для создания новой серии необходимо в редакторе перейти на главную страницу **Chart,** а на ней открыть страницу **Series** (рис. 5.6). На этой странице нужно щелкнуть на кнопке **Add**, а затем в появившемся диалоге выбрать тип серии. После этого в списке на странице **Series** появляется строка новой серии. Здесь можно переопределить тип, цвет и видимость серии, щелкнув на соответствующей зоне строки.

Все остальные страницы блокнота на главной странице **Chart** предназначены для настройки параметров графика.

Теперь необходимо перейти на главную страницу **Series** и на ней из списка названий серий выбрать необходимую. После этого на странице **Data Source** из списка выбирается строка **DataSet.** Далее в появившемся списке **DataSet** выбирается нужный набор данных.

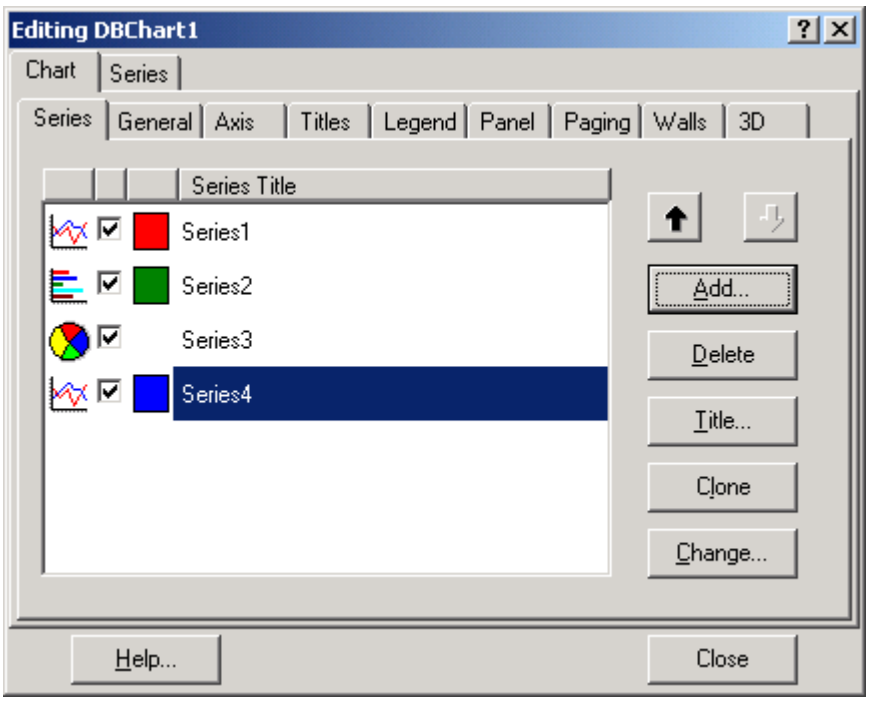


Рисунок 5.6. Специализированный редактор компонента TDBChart

Список X позволяет выбрать поле набора данных, значения которого будут последовательно откладываться по оси абсцисс. Список Y позволяет выбрать поле набора данных, значения которого будут отложены по оси ординат. Соответствие между значениями полей по двум осям определяется принадлежностью к одной записи набора данных. Выбор поля в списке **Labels** привязывает его значения в виде меток к оси абсцисс.

**Примечание**

*Здесь описан набор элементов управления для линейного типа серии. Для других типов элементы управления могут отличаться.*

Теперь осталось только открыть набор данных и компонент TDBChart построит график.

Аналогичным образом на этот же компонент можно поместить и другие графики.

Таблица 5.8. Свойства и методы компонента TDBChart

| **Объявление** | **Описание** |
|---|---|
| **Свойства** | |
| property AutoRefresh : Boolean; | Разрешает или запрещает обновление данных в серии при открытии связанного набора данных |
| property Refreshlnterval: Longlnt; | Задает временной интервал в секундах между обновлениями данных в сериях из связанных наборов данных |
| property ShowGlassCursor: Boolean; | Разрешает показ курсора "песочные часы" при обновлении данных |
| **Методы** | |
| procedure CheckDataSource; | Обновляет данные в сериях |
| function IsValidData-Source (ASeries : TChartSeries; ACompo-nent: TComponent) : Boolean; virtual; | Проверяет, связан ли набор данных AComponent с серией ASeries. В случае успеха проверки возвращает True |
| procedure RefreshData; | Обновляет данные во всех сериях |
| procedure Ref reshDa-taSet (ADataSet : TDataSet; ASeries: TChartSeries); | Считывает все записи в наборе данных AdataSet и переносит их в серию ASeries |
| **Методы-обработчики событий** | |
| property OnProcessRecord : | Вызывается при переносе данных из отдельной записи набора данных в серию |

| | |
|---|---|
| TProcessRecordEvent; | |

Компоненты отображения данных играют важную роль при создании интерфейсов приложений баз данных. Разнообразие предлагаемых элементов управления позволяет решать любые задачи по организации взаимодействия пользователя с базой данных. Все они взаимодействуют с набором данных через компонент TDataSource.

**Форма контроля: контрольные воспросы**

1. Поясните классификацию компонентов отображения данных.
2. Назовите общие свойства компонентов отображения данных.
3. Табличное представление данных.
4. Навигация по набору данных.
5. Объясните назначение кнопок компонента TDBNavigator.
6. Представление отдельных полей.
7. Синхронный просмотр данных.
8. Механизм синхронного просмотра.
9. Графическое представление данных.

**Рекомендуемая литература**

**Основная литература**

1. Брукс Ф. П. Как проектируются и создаются программные комплексы. М.: Наука, 2009.

2. Майерс Г. Искусство тестирования программ.-М.: Финансы и статистика, 1982.

3. Фокс Дж. Программное обеспечение и его разработка.-М.: Мир, 1985.

4. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2007 г.

5. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.

6. Федоров А., Елманова Н. Введение в базы данных. Настольные СУБД – КомпьютерПресс, 2000, №4.

**Дополнительная литература**

7. С.Бобровский. Delphi 7. Учебный курс. Издательство «ПИТЕР»,2005г.

8. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».

## Тема № 6
## Процессор баз данных Borland Database Engine

**План:**

1. Архитектура и функции BDE
2. Псевдонимы баз данных и настройка BDE
3. Интерфейс прикладного программирования BDE
4. Соединение с источником данных
5. Компоненты доступа к данным
6. Компонент TTable

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

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

и т. д.

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

Одним из традиционных способов взаимодействия приложения, созданного в среде разработки Delphi, и базы данных является использование процессора баз данных Borland Database Engine 5. Он представляет собой набор динамических библиотек, функции которых позволяют не только обращаться к данным, но и эффективно управлять ими на стороне приложения.

Для работы с источниками данных при посредстве BDE в Delphi имеется специальный набор компонентов, расположенных на странице BDE Палитры компонентов. Эти компоненты для работы с базами данных используют возможности BDE, обращаясь к его функциям и процедурам. Механизм доступа к BDE инкапсулирован в базовом классе TBDEDataSet. Поэтому в процессе программирования у вас не будет необходимости использовать функции BDE напрямую. Почти все, что можно сделать путем прямого обращения, можно сделать и через компоненты — это проще и надежнее.

Тем не менее внутреннюю организацию механизма доступа к данным всегда полезно знать. Кроме этого, всегда полезно знать и уметь использовать дополнительные возможности, которые BDE может предоставить разработчику.

BDE взаимодействует с базами данных при посредстве драйверов. Для особенно распространенных локальных СУБД разработан набор стандартных драйверов. Работа с наиболее распространенными серверами БД осуществляется при помощи драйверов системы SQL Links. Кроме этого, если для базы данных существует драйвер ODBC, то можно использовать и его. Достаточно зарегистрировать этот драйвер в BDE.

Однако BDE не претендует на всеобъемлющую универсальность и имеет некоторые недостатки. Это, например, снижение скорости работы приложения, недостатки реализации некоторых драйверов и т. д. В документации к Delphi 7 содержится предупреждение, что после 2002 года фирма Borland перестанет поддерживать BDE и рекомендует использовать технологию dbExpress, которая также рассматривается в настоящей книге.

**Архитектура и функции BDE**

BDE представляет собой набор динамических библиотек, которые "умеют" передавать запросы на получение или модификацию данных из приложения в нужную базу данных и возвращать результат обработки. В процессе работы библиотеки используют вспомогательные файлы языковой поддержки и информацию о настройках среды.

В составе BDE поставляются стандартные драйверы, обеспечивающие доступ к СУБД Paradox, dBASE, FoxPro и текстовым файлам. Локальные драйверы (рис. 6.1) устанавливаются автоматически совместно с ядром процессора. Один из них можно выбрать в качестве стандартного драйвера, который имеет дополнительные настройки, влияющие на функционирование процессора БД.

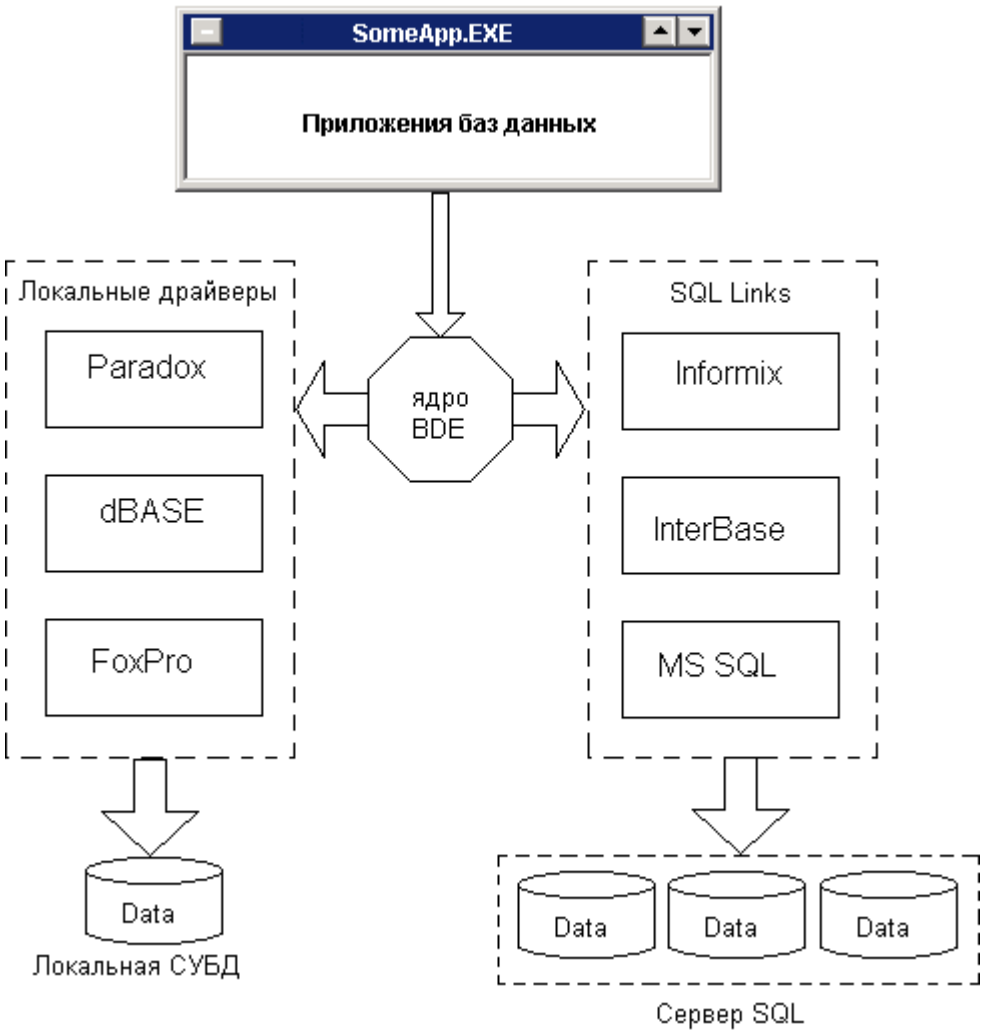


Рисунок 6.1. Структура процессора баз данных BOE

Доступ к данным серверов SQL обеспечивает отдельная система драйверов — SQL Links. С их помощью в Delphi можно без особых проблем разрабатывать приложения для серверов Oracle 8, Informix, Sybase, DB2 и, естественно, InterBase. Эти драйверы необходимо устанавливать дополнительно.

Помимо этого, в BDE имеется очень простой механизм подключения любых драйверов ODBC (к примеру, Microsoft Access) и создания на их основе сокетов ODBC.

**Примечание** *С точки зрения пользователя процесс подключения локального драйвера и драйвера SQL Links практически не отличается, за исключением деталей настройки. Настройка драйверов и собственных параметров BDE осуществляется при помощи специальной утилиты — BDE Administrator и рассматривается далее в этой главе.*

В состав BDE входят следующие функциональные подсистемы.

- Администратор системных ресурсов управляет процессом подключения к данным — при необходимости устанавливает нужные драйверы, а при завершении работы автоматически освобождает занятые ресурсы. Поэтому BDE всегда использует ровно столько ресурсов, сколько необходимо.
- Система обработки запросов обеспечивает выполнение запросов SQL или QBE от приложения к любым базам данных, для которых установлен драйвер, даже если сама СУБД не поддерживает прямое использование запросов SQL.
- Система сортировки является запатентованной технологией и обеспечивает очень быстрый поиск по запросам SQL и через стандартные драйверы аля Paradox и dBASE.
- Система пакетной обработки представляет собой механизм преобразования данных из одного формата в другой при выполнении операций над целыми таблицами. Эта система использована в качестве основы для компонента TBatcMove и утилиты DataPump (автоматического переноса структур данных между базами данных), входящей в стандартную поставку BDE.
- Менеджер буфера управляет единой для всех драйверов буферной областью памяти, которую одновременно могут использовать несколько драйверов. Это позволяет существенно экономить системные ресурсы.
- Менеджер памяти взаимодействует с ОС и обеспечивает эффективное использование выделяемой памяти. Ускоряет работу драйверов, которые для получения небольших фрагментов памяти обращаются к нему, а не к ОС. Дело в том, что менеджер памяти выделяет большие объемы оперативной памяти и затем распределяет ее небольшими кусками между драйверами согласно их потребностям.
- Транслятор данных обеспечивает преобразование форматов данных для различных типов БД.
- Кэш BLOB используется для ускорения работы с данными в формате BLOB.
- SQL-генератор транслирует запросы в формате QBE в запросы SQL.
- Система реструктуризации обеспечивает преобразование наборов данных в таблицы Paradox или dBASE.

- Система поддержки драйверов SQL повышает эффективность механизма поиска при выполнении запросов SQL.
- Таблицы в памяти. Этот механизм позволяет создавать таблицы непосредственно в оперативной памяти. Используется для ускорения обработки больших массивов данных, сортировки, преобразования форматов данных.
- Связанные курсоры обеспечивают низкоуровневое выполнение межтабличных соединений. Позволяют разработчику не задумываться над реализацией подобных связей при работе на уровне VCL — для этого достаточно установить значения нескольких свойств.
- Менеджер конфигурации обеспечивает разработчику доступ к информации о конфигурации драйверов.

Перечисленные функции реализованы в динамических библиотеках, которые, собственно, и называются процессором БД (табл. 6.1).

Таблица 6.1. Ядро процессора баз данных BOE 5

| **Имя файла** | **Назначение** |
|---|---|
| IDAPI32.DLL | Базовая динамическая библиотека BOE |
| IDPROV.DLL | Динамическая библиотека, отвечающая за работу серверной части приложения |
| BLW32.DLL | Динамическая библиотека, обеспечивающая поддержку драйверов национальных языков |
| IDBAT32.DLL | Динамическая библиотека с функциями межтабличного переноса данных |
| IDQBE32.DLL | Динамическая библиотека, обеспечивающая работу запросов по примеру (Query By Example, QBE) |
| IDSQL32.DLL | Динамическая библиотека, обеспечивающая обработку запросов SQL |
| IDASCI32.DLL | Динамическая библиотека, обеспечивающая работу драйвера текстовых файлов |
| IDPDX32.DLL | Динамическая библиотека, обеспечивающая работу драйвера Paradox |
| IDDBAS32.DLL | Динамическая библиотека, обеспечивающая работу драйвера dBASE |
| DODBC32.DLL | Динамическая библиотека, обеспечивающая работу драйвера сокета ODBC |
| IDR20009.DLL | Динамическая библиотека ресурсов, содержащая сообщения об ошибках |
| IDDA032.DLL | Динамическая библиотека, обеспечивающая работу драйверов Microsoft Access 95 и Jet Engine 3.0 |
| IDDA3532.DLL | Динамическая библиотека, обеспечивающая работу драйверов Microsoft Access 97 и Jet Engine 3.5 |

| | |
|---|---|
| IDDR32.DLL | Динамическая библиотека для работы с Репозиторием данных |

Кроме этого имеется шесть дополнительных DLL, обеспечивающих работу BDE с серверами Oracle и Microsoft SQL Server.

Псевдонимы баз данных и настройка BDE

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

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

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

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

Для управления псевдонимами баз данных, настройки стандартных и дополнительных драйверов в составе BDE имеется специальная утилита - BDE Administrator (исполняемый файл BDEADMIN.EXE). Стандартная конфигурация BDE сохраняется в файле IDAPI.CFG. При необходимости текущую конфигурацию можно сохранить в новом файле с расширением cfg или загрузить заново при помощи команд **Save As Configuration и Open Configuration** из меню **Object.**

В верхней части окна утилиты расположена Панель инструментов, кнопки которой используются при работе с конкретным элементом настройки BDE. Рабочая область утилиты BDE Administrator представляет собой двухстраничный блокнот.

Страница **Databases** (рис. 6.2) содержит иерархическое дерево, в узлах которого расположены установленные в системе на данный момент псевдонимы БД. При выборе какого-либо псевдонима в правой части панели появляется путь к файлам базы данных и перечень параметров драйвера, соответствующего псевдониму, которые можно настраивать вручную.

Страница **Configuration** (рис. 6.3) используется для настройки параметров драйверов BDE, предназначенных для обеспечения доступа к локальным СУБД и серверам БД. Также здесь определяется системная конфигурация BDE, которая включает параметры числовых форматов, форматов даты и времени. Вся информация на этой странице также структурирована в виде иерархического дерева.

При выборе в левой части панели утилиты какого-либо узла, в правой части на странице **Definition** отображается вся необходимая информация для этого объекта.

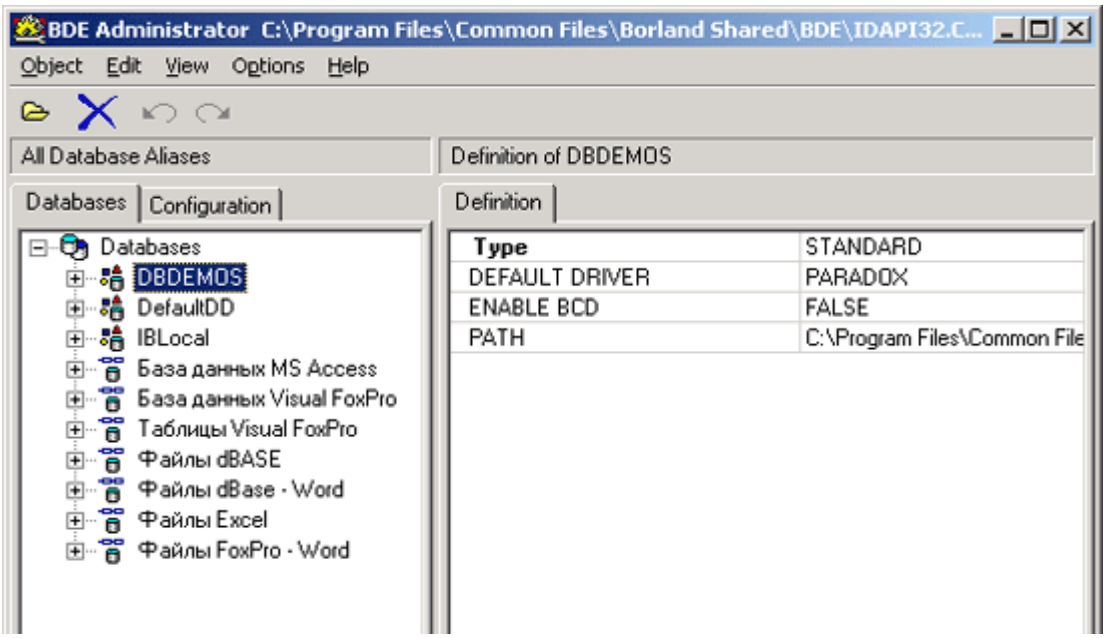


Рисунок 16.2. Окно утилиты BDE Administrator с открытой страницей Databases

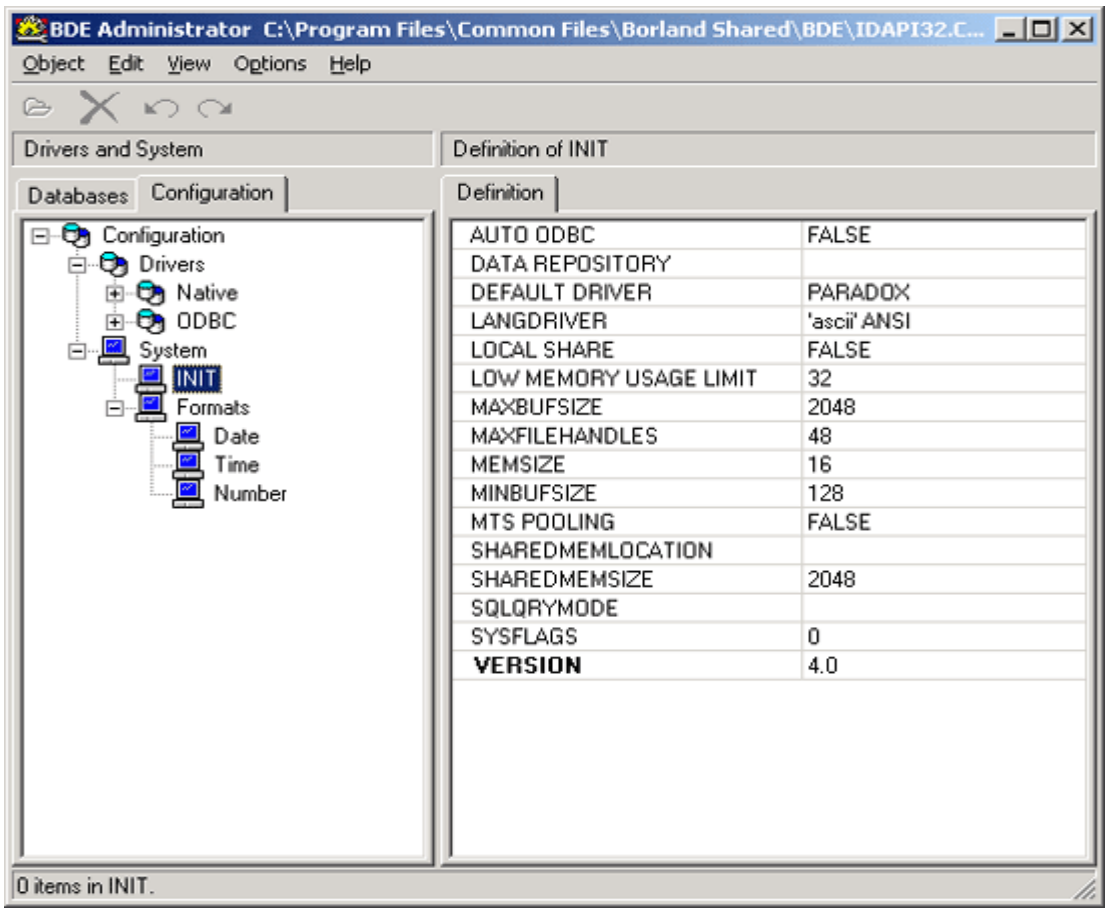


Рисунок 6.3. Окно утилиты BDE Administrator с открытой страницей Configuration

Сохранение изменений осуществляется при помощи команд меню **Object,** всплывающего меню или при перемещении на другой псевдоним.

Для создания нового псевдонима требуется выбрать команду **New** из меню **Object** или из всплывающего меню узла **Databases** на одноименной странице. Затем в появившемся простом диалоге задается необходимый драйвер.

Отметим, что один из четырех стандартных локальных драйверов устанавливается на странице **Configuration** в качестве предопределенного, поэтому в списке он доступен под названием STANDARD, а остальные не видны вообще. Из драйверов SQL Links доступны те, которые были установлены при инсталляции Delphi или позже.

Кроме того, в списке можно выбрать один из драйверов ODBC, установка которых осуществляется стандартными системными средствами на Панели управления Windows.

После выбора драйвера в дереве псевдонимов БД появляется новый узел, для драйвера которого требуется установить необходимые параметры (см. ниже).

Для четырех локальных драйверов список параметров в правой части панели утилиты на странице **Definition** ограничивается параметрами стандартного драйвера (STANDARD), подробная настройка для каждого драйвера осуществляется на странице **Configuration.**

Назначение параметров локальных драйверов BDE (Paradox, dBASE, FoxPro, ASCII) представлено в табл. 6.2.

Таблица 6.2. Параметры драйверов BDE для локальных баз данных

| Параметр | Назначение |
|---|---|
| **STANDARD** | |
| DEFAULT DRIVER | Задает тип конкретного локального драйвера (Paradox, dBASE, FoxPro, ASCII) |
| ENABLE BCD | Определяет способ представления вещественных чисел. При значении True такие числа преобразуются в формат BCD (Binary Coded Decimals— десятичные с двоичным кодированием). Точность составляет 20 знаков после запятой |
| PATH | Указывает путь к файлам базы данных |
| **PARADOX** | |
| NET DIR | Указывает путь к файлу обеспечения сетевого доступа к БД PDOXUSRS.NET. Драйвер приложения, которое работает с БД локально, должен указывать на этот файл, расположенный на том же компьютере. Драйвер приложения, обращающегося к БД по сети, должен указывать на подключенный сетевой диск с этим файлом |
| VERSION | Нередактируемая информация о версии драйвера |
| TYPE | Тип СУБД. Для Paradox имеет значение FILE. Только для чтения |
| LANGDRIVER | Определяет драйвер языковой поддержки (используйте драйвер Paradox Cyrr 866) |
| BLOCK SIZE | Задает размер блоков дискового пространства для хранения записей, кратно 1024 |
| FILL FACTOR | Определяет процент заполнения блока дискового пространства при хранении индексов, по умолчанию 95% |
| LEVEL | Задает формат временной таблицы в памяти:<br>• 3 — совместим с Paradox 3.5 и ниже;<br>• 4 — Paradox 4.0;<br>• 5 — Paradox 5.0;<br>• 7 - Paradox для WIN32 |
| STRICTINTEGRTY | Определяет возможность использования приложениями на базе Paradox 4.0 более поздних таблиц со ссылочной целостностью. При значении True использование разрешается, но возникает риск нарушения целостности данных |
| **DBASE** | |
| VERSION | Нередактируемая информация о версии драйвера |
| TYPE | Тип СУБД. Для dBASE имеет значение FILE. Только для чтения |

| | |
|---|---|
| LANGDRIVER | Определяет драйвер языковой поддержки (используйте драйвер dBASE RUS cp866) |
| LEVEL | Задает формат таблиц. Значение соответствует номеру версии СУБД |
| MDX BLOCK SIZE | Размер блоков для файлов с расширением mdx, кратно 51 2 |
| MEMO FILE BLOCK SIZE | Размер блоков для файлов с данными типа Memo (расширение dbt), кратно 512 |
| **FOXPRO** | |
| VERSION | Нередактируемая информация о версии драйвера |
| TYPE | Тип СУБД. Для FoxPro имеет значение FILE. Только для чтения |
| LANGDRIVER | Определяет драйвер языковой поддержки |
| LEVEL | Имеет значение 25 |

**Примечание**

*Драйвер текстовых файлов ASCIIDRV имеет параметры стандартного драйвера.*

Назначение параметров драйверов SQL Links для серверов SQL представлено в табл. 6.3. Сначала приведены параметры, которые встречаются в двух и более драйверах, затем уникальные для каждого драйвера параметры. Драйверы для серверов InterBase и Sybase не представлены, т. к. содержат только общие для двух серверов параметры.

Таблица 6.3. Параметры драйверов BOE для серверов SQL

| **Параметр** | **Назначение** |
|---|---|
| **Общие параметры (встречаются как минимум у двух драйверов)** | |
| VERSION | Нередактируемая информация о версии драйвера |
| TYPE | Тип СУБД. Только для чтения |
| DLL | Название библиотеки динамического связывания SQL Links для 16-разрядного драйвера. Только для чтения |
| DLL32 | Название библиотеки динамического связывания SQL Links для 32-разрядного драйвера. Только для чтения |
| DRIVER FLAGS | Используется только при необходимости применения старых версий драйвера, где не поддерживается уровень изоляции транзакций Read Committed. Для этого необходимо установить значение 512 |
| TRACE MODE | Содержит битовую маску, которая определяет тип выдаваемой отладочной информации |
| BATCH COUNT | Задает число записей, модифицируемых в одном пакете при фиксации транзакций |

| | |
|---|---|
| BLOB SIZE | Размер кэша для данных типа BLOB. Диапазон от 32К до 1000К |
| BLOBS TO CACHE | Задает число кэшируемых записей с данными BLOB. Диапазон от 64 до 65 536 |
| ENABLE BCD | Определяет способ представления вещественных чисел. При значении True такие числа преобразуются в формат BCD (Binary Coded Decimals — десятичные с двоичным кодированием), который позволяет округлять погрешности высших разрядов дробной части числа. Изменение параметра для псевдонима работает, только если параметр драйвера на странице Configuration не пустой |
| ENABLE SCHEMA CACHE | Определяет режим кэширования структуры данных. При значении True структура таблиц БД кэшируется локально в каталоге, задаваемом параметром SCHEMA CACHE DIR. Рекомендуется использовать только для баз данных с постоянной структурой |
| LANGDRIVER | Определяет драйвер языковой поддержки |
| MAX ROWS | Ограничивает максимальное число записей, которое может быть передано клиенту в ответ на запрос. Значение по умолчанию (ограничений нет) |
| OPEN MODE | Режим работы с записями БД:<br>• READ/WRITE — полный доступ;<br>• READ ONLY — только чтение |
| SCHEMA CACHE DIR | Каталог для локального кэширования структуры данных (см. параметр ENABLE SCHEMA CACHE) |
| SCHEMA CACHE SIZE | Задает число таблиц, структура данных которых может кэшироваться |
| SCHEMA CACHE TIME | Задает время хранения кэшируемой структуры данных:<br>• -1 — время не ограничено;<br>• 0 — данные не кэшируются;<br>• 1 — 21 47483647 — секунды |
| SERVER NAME | Указывает путь к таблицам БД (это может быть локальный маршрут или маршрут с указанием удаленного сервера БД) |
| SQLPASSTHRU MODE | Задает способ использования соединения с сервером прямыми запросами SQL и запросами, управляемыми пользователем.<br>SHARED AUTOCOMMIT— соединение используется совместно и прямые запросы фиксируются автоматически.<br>SHARED NO AUTOCOMMIT— соединение используется совместно и прямые запросы фиксируются сервером самостоятельно.<br>NOT SHARED — совместное использование запрещено |

| | |
|---|---|
| | |
| SQLQRYMODE | Задает режим управления запросами.<br>NULL — сначала запрос передается серверу, если тот не может обработать его, запрос выполняется локально.<br>SERVER — запрос передается серверу.<br>LOCAL — запрос выполняется локально |
| VENDOR INIT | Название файла динамической библиотеки поставщика |
| CONNECT TIMEOUT | Определяет временной интервал, после которого клиент попытается восстановить прерванную связь с сервером |
| TIMEOUT | Задает время ожидания ответа сервера на запрос |
| BLOB EDIT LOGGING | Управляет механизмом сохранения всех изменений для полей типа BLOB. При значении True изменения сохраняются |
| DATABASE NAME | Имя базы данных |
| MAX QUERY TIME | Задает максимальное время ожидания ответа на запрос |
| USER NAME | Имя пользователя, которое используется сервером при подключении |
| **Microsoft SQL Server (MSSSQL)** | |
| MAX DBPROCESSES | Максимальное число процессов, одновременно работающих в данном соединении |
| APPLICATION NAME | Имя приложения, помогающее серверу идентифицировать процессы |
| DATE MODE | Определяет формат даты:<br>• 0 - МДГ;<br>• 1 - ДМГ; .<br>• 2-ГМД |
| HOST NAME | Содержит имя рабочей станции. Помогает серверу при идентификации процессов |
| NATIONAL LANG NAME | Задает национальный язык, который используется для вывода текста в сообщениях об ошибках |
| TDS PACKET SIZE | Определяет размер пакетов потоков данных |
| **Oracle (ORACLE)** | |

| | |
|---|---|
| NET PROTOCOL | Устанавливает сетевой протокол передачи данных |
| **Informix (INFORMIX)** | |
| DATE SEPARATOR | Задает разделитель для формата даты |
| **Microsoft Access (MSACCIESS)** | |
| SYSTEM DATABASE | Путь к системной базе данных с информацией о правах доступа. При изменении параметра драйвер необходимо перезагрузить |
| **DB2 (DB2)** | |
| DB2 DSN | Задает имя соединения с БД. Это название псевдонима клиента DB2, который создается на сервере |
| DRIVER | Имя драйвера DB2 |
| ROWSET SIZE | Определяет число записей, передаваемых одновременно |
| **Драйверы ODBC** | |
| ODBC DRIVER | Имя драйвера ODBC |
| ODBC DSN | Имя набора данных ODBC |

После настройки параметров драйвера и сохранения текущей конфигурации новый псевдоним становится доступен для любого приложения, использующего BDE.

Страница **Configuration,** помимо настройки установленных в BDE драйверов, позволяет редактировать параметры, используемые BDE при инициализации приложения. Эти параметры доступны при выборе узлов **System,** а затем **INIT** иерархического дерева. Назначение параметров представлено в табл. 6.4.

Таблица 6.4. Параметры инициализации приложения

| **Параметр** | **Назначение** |
|---|---|
| AUTO ODBC | В значении True при каждой инициализации в BDE автоматически импортируются все установленные в системе драйверы ODBC |
| DATA REPOSITORY | Имя текущего словаря данных |
| DEFAULT DRIVER | Локальный драйвер, используемый по умолчанию в драйвере STANDARD |
| LANGDRIVER | Драйвер языковой поддержки. При использовании стандартных локальных драйверов это значение перекрывают те, которые определены непосредственно в конфигурациях драйверов Paradox, dBASE, FoxPro, ASCII |

| | |
|---|---|
| LOCAL SHARE | Устанавливает режим совместного использования файлов приложениями, работающих через BDE и другие программы. В значении True совместное использование разрешено |
| LOW MEMORY USAGE LIMIT | Максимальный объем памяти (в Кбайтах), который BDE пытается использовать в первом Мбайте оперативной памяти |
| MAXBUFSIZE | Максимальный размер кэша данных. Он должен быть не меньше значения параметра MINBUFSIZE и кратен 1 28 |
| MAXFILEHANDLES | Максимальное число используемых файлов |
| MEMSIZE | Максимальный объем используемой BDE памяти в Мбайтах |
| MINBUFSIZE | Минимальный размер кэша данных. Он должен быть не больше значения параметра MAXBUFSIZE и кратен 128 |
| MTS POOLING | Управляет режимом объединения ресурсов MTS (Microsoft Transaction Server). Обеспечивает лучшую производительность |
| SHAREDMEMLOCATION | Содержит адрес памяти, который пытаются использовать Менеджер памяти и Менеджер буфера. При возникновении конфликтов адрес необходимо поменять вручную. Задается только второе слово адреса |
| SHAREDMEMSIZE | Максимальный объем памяти, используемый Менеджером памяти и Менеджером буфера |
| SQLQRYMODE | См. табл. 16.2 |
| SYSFLAGS | Не используется |
| VERSION | Номер внутренней версии BDE. Только для чтения |

Также на странице **Configuration** устанавливаются параметры форматов даты, времени и чисел. Доступ к параметрам осуществляется через узлы **System и Format.**

Интерфейс прикладного программирования BDE

Как уже говорилось выше, любое приложение Delphi, работающее с базами данных и написанное с использованием стандартных компонентов доступа к данным, обращается к данным и получает результат при помощи BDE. При этом механизм доступа к данным использует вызовы функций из API BDE.

Достаточно сложно представить себе такую ситуацию, когда возникает необходимость создания приложения, использующего только функции BDE, без применения компонентов доступа к данным VCL. А вот отдельные функции вполне могут понадобиться в любой программе. Поэтому рассмотрим процесс

работы приложения, использующего вызовы BDE, т. к. это дает хорошую возможность понять механизм доступа к данным, который реализован в Delphi.

Итак, для создания приложения на основе вызовов функций BDE необходимо выполнить следующие операции:

1. Инициализация BDE (функция DbiInit).

2. Открытие объекта базы данных (функция DbiOpenDatabase).

3. Определение рабочего каталога (функция obiSetDirectory), если на предыдущем этапе не задается псевдоним БД.

4. Определение временного каталога (функция DbiSetPrivateoir).

5. Открытие набора данных и создание курсора (функции DbOpenTable, DbiQExec и пр.; дескриптор курсора hDBICur).

6. Заполнение структуры cuRProps, содержащей данные о курсоре и наборе данных **(**функция DbiGetCursorProps).

7. Выделение памяти для буфера записи.

8. Навигация набору данных (функции DbiSetToBegin, DbiSetToEnd, DbiSetToCursor и пр.)

9. Чтение необходимой записи **(**функции DbiGetRelativeRecord, DbiGetNextRecord, DbiGetRecord, DbiGetPriorRecord и др.).

10. Чтение или обновление необходимого поля (функции DMGetFieid, DbiPutField).

11. Освобождение всех ресурсов (освобождение буфера записи, закрытие курсора, таблицы, BDE).

При использовании в программе функций из API BDE необходимо включить в секцию uses модуль BDE.

В прикладном программировании задачи, которые требовали бы выполнения всех описанных выше операций, практически не встречаются. Между тем, включение в программный код отдельных функций API BDE оправдано. В качестве примера рассмотрим приложение, которое позволяет очистить таблицу базы данных (рис. 6.4).

Дело в том, что при удалении записи из таблицы локальной СУБД (например, Paradox) размер файла таблицы остается прежним, даже если удалить все записи. То есть на самом деле запись не удаляется, а только становится недоступной. При интенсивном использовании базы данных файлы таблиц могут занимать значительные объемы дискового пространства при довольно умеренном числе записей.

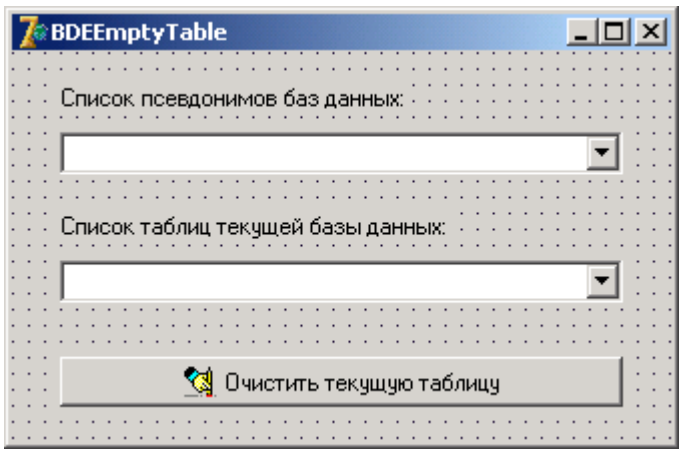


Рисунок 6.4. Главная форма проекта BDEEmptyTable

Полная очистка таблиц базы данных осуществляется функцией DbiErr.ptyTable из API BDE. Именно она используется в демонстрационном приложении для радикального уменьшения размера таблиц.

**Примечание**

*Функция DbiEmptyTable используется в методе EmptyTable компонентов доступа к данным.*

При открытии главной формы (метод-обработчик FormShow) функция Dbiinit осуществляет инициализацию BDE. Затем функция DbiOpenDatabaseList создает в памяти временную таблицу, в которую записываются характеристики каждой зарегистрированной базы данных. Для этого применяется структура DBDesc. Курсор hcursor обеспечивает доступ к записям о базах данных.

После этого функция DbiGetNextRecord позволяет осуществить последовательное считывание имен псевдонимов баз данных (для этого в параметре передается указатель на структуру DBDesc) и их запись в список компонента AliasesList типа TComboBox.

При выборе из этого списка конкретного псевдонима работает метод-обработчик AliasesListchange. В нем открывается соответствующая база данных (функция DbiOpenDatabase), доступ к которой в дальнейшем осуществляется через дескриптор hDB.

Функция DbiopenTableList создает временную таблицу в памяти, в которую помещаются данные о таблицах выбранной базы данных в соответствии с форматом структуры TBLBaseDesc. Функция DbiGetNextRecord позволяет передать эту информацию в список компонента TablesList типа TComboBox.

При щелчке на кнопке EmptyBtn в методе-обработчике EmptyBtndick работает функция DbiEmptyTabie, которая очищает выбранную ранее в компоненте TablesList таблицу.

Теперь рассмотрим пример простейшего приложения, которое может отображать два поля из таблицы COUNTRY. DB в демонстрационной базе данных DBDEMOS. Эта база данных поставляется в комплекте Delphi. В примере использованы только функции API BDE.

Проект называется DirectBDE и имеет только одну форму, в которой отображаются сведения из таблицы COUNTRY. DB о государствах и их столицах

(рис. 6.5). Кнопки в нижней части формы позволяют перемещаться по набору данных.

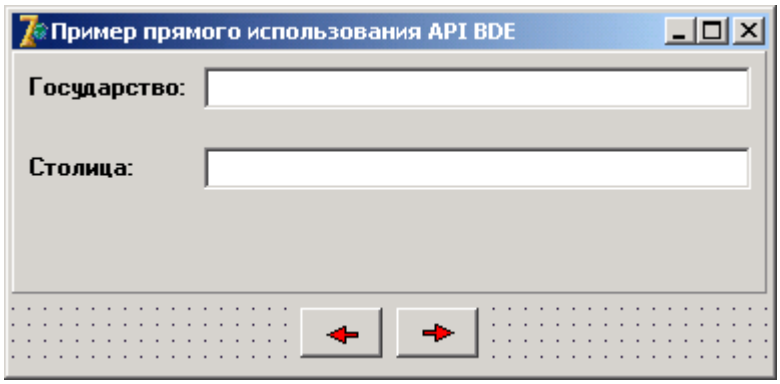


Рисунок 6.5. Главная форма проекта DirectBDE

При показе главной формы приложения в процедуре Formshow проводится инициализация BDE, открытие базы данных и таблицы. При этом создаются дескрипторы базы данных hDB и курсора таблицы hour, которые играют в дальнейшей работе приложения важную роль. Если при создании базы данных не указывать псевдоним БД, то обязательно нужно определить рабочий каталог базы данных с помощью функции DbiSetoirectory.

После этого отводится память под буфер записи, в который будут передаваться значения полей текущей строки таблицы. Размер буфера определяется при помощи Структуры CURPropS.

Затем курсор устанавливается на начало набора данных и на первую запись и осуществляется чтение значений двух полей таблицы.

Навигация по набору данных реализована в методах-обработчиках на нажатие кнопок формы. Их действие аналогично за исключением направления перемещения. При щелчке на кнопке выполняется переход на следующую или предыдущую запись, данные из новой записи помещаются в буфер записи RecBuf. Оттуда при помощи функции DMGetField осуществляется чтение значений полей. При достижении начала или конца набора данных кнопка деактивируется.

При закрытии формы проводятся операции по освобождению памяти буфера записи, закрытию базы данных и BDE.

Соединение с источником данных

Все обращения из приложения к таблицам одной базы данных осуществляются через одно соединение, на которое замыкаются все компоненты доступа к данным, имеющие соответствующие значения свойства DatabaseName (см. ниже).

Все управление одиночным соединением с какой-либо базой данных в BDE осуществляется компонентом TDatabase (табл. 6.5). В процессе работы компонент активно использует параметры псевдонимов и драйверов BDE.

Таблица 6.5. Свойства и методы компонента TDatabase

| **Объявление** | **Тип** | **Описание** |
|---|---|---|

| **Свойства** | | |
|---|---|---|
| property AliasName: string; | Pb | Задает имя псевдонима BDE используемой базы данных |
| property Connected: Boolean; | Pb | Управляет включением соединения с базой данных |
| property DatabaseName: string; | Pb | Определяет имя базы данных |
| property DataSetCount: Integer; | Ro | Возвращает число открытых наборов данных, работающих через данное соединение |
| property DataSets [Index: Integer]: TDBDataSet; | Ro | Индексированный список всех объектов открытых наборов данных этого соединения |
| property Directory: string; | Pu | Определяет текущий каталог для баз данных Paradox и dBASE |
| property DriverName: string; | Pb | Содержит имя драйвера базы данных |
| property Exclusive: Boolean; | Pb | При значении True другие приложения не могут работать с базой данных одновременно с этим компонентом |
| type HDBIDB: Longint; property Handle: HDBIDB; | Pu | Дескриптор BDE. Используется для прямых вызовов функций API BDE |
| property HandleShared: Boolean; | Pu | При значении True дескриптор BDE компонента доступен в компоненте TSession |
| property InTransaction: Boolean | Ro | Показывает состояние транзакции. При значении True транзакция выполняется |
| property IsSQLBased: Boolean; | Ro | При значении True соединение работает через драйвер SQL Links |
| property KeepConnection: Boolean; | Pb | При значении True соединение продолжает оставаться активным после закрытия всех наборов данных. При значении False после закрытия последнего набора данных соединение закрывается |
| type TLocale: Pointer; property Locale: TLocale; | Ro | Указывает на языковый драйвер BDE, используемый при работе с базой данных |
| property LoginPrompt: Boolean; | Pb | Управляет отображением стандартного диалога регистрации пользователя при подключении к серверу |

| | | |
|---|---|---|
| property Params: TStrings; | Pb | Содержит список значений параметров псевдонима BDE, которые пользователь задает перед подключением к серверу |
| property Session: TSession | Ro | Указывает на компонент TSession, который управляет работой данного компонента |
| property SessionAlias: Boolean; | Ro | При значении True при подключении к БД используется псевдоним сессии |
| property SessionName: string; | Pb | Содержит имя сеанса, который управляет работой компонента |
| property Readonly: Boolean; | Pb | Управляет режимом доступа к данным "только для чтения" |
| property Temporary: Boolean; | Pu | Значение True говорит о том, что экземпляр компонента создан во время выполнения |
| type TTraceFlag = (tfQPrepare, tfQExecute, tfError, tfStmt, tf Connect, tfTransact, tfBlob, tfMisc, tfVendor, tfDataln, tfDataOut) ; TTraceFlags = set of TTraceFlag; property TraceFlags: TTraceFlags; | Pu | Определяет перечень операций, выполнение которых отображается в утилите SQL Monitor при выполнении приложения |
| type TTransIsolation = (tiDirtyRead, tiReadCommitted, tiRepeatableRead) ; property Translsolation: TTransIsolation; | Pb | Определяет уровень изоляции транзакций:<br>• tiDirtyRead— незавершенное чтение;<br>• tiReadCommitted — завершенное чтение;<br>• tiRepeatableRead — повторяемое чтение |
| **Методы** | | |
| procedure ApplyUpdates (const DataSets: array of TDBDataSet); | Pu | Фиксирует все изменения в наборах данных, работающих через это соединение, в базе данных |
| procedure Close; | Pu | Закрывает все открытые наборы данных и соединение |
| procedure CloseDatasets; | Pu | Закрывает все открытые наборы данных, работающие через это соединение |

| | | |
|---|---|---|
| procedure Commit; | Pu | Завершает выполнение текущей транзакции и фиксирует все изменения в базе данных |
| function Execute (const SQL: string; Params : TParams = nil; Cache: Boolean = False; Cursor: phDBICur = nil) : Integer; | Pu | Выполняет запрос SQL без использования компонента TQuery. Текст запроса содержится в параметре SQL. Параметры запроса определяются параметром Params. Режим кэширования изменений включается параметром Cache. Параметр Cursor может использоваться при работе с функциями BDE, использующими курсор набора данных (см. гл. 14) |
| procedure FlushSchemaCache (const TableName: string); | Pu | Изменяет представление о структуре таблиц БД, загруженной в память |
| procedure Open; | Pu | Открывает соединение |
| procedure Rollback; | Pu | Отменяет все операции текущей транзакции и завершает ее |
| procedure StartTransaction; | Pu | Начинает выполнение транзакции |
| procedure ValidateName (const Name: string) ; | Pu | Вызывает исключительную ситуацию, если база данных Name уже открыта в текущей сессии |
| **Методы-обработчики событий** | | |
| type TLoginEvent = procedure (Database: TDatabase; LoginParams: TStrings) of object;<br>property OnLogin: TLoginEvent; | Pb | Вызывается при регистрации пользователя на сервере |
| property AfterConnect: TNotifyEvent; | Pb | Вызывается после подключения |
| property AfterDisconnect: TNotifyEvent; | Pb | Вызывается после отключения |
| property BeforeConnect: TNotifyEvent; | Pb | Вызывается перед подключением |
| property AfterDisconnect: TNotifyEvent; | Pb | Вызывается перед отключением |

Обычно компонент TDatabase размещается в модуле данных приложения.

Для определения базы данных (сервера), с которой приложение устанавливает соединение при помощи компонента TDatabase, чаще используется

свойство AliasName. Свойства DatabaseName и DriverName предоставляют альтернативный способ создания соединения.

Если соединение задано свойством AliasName,то свойство DatabaseName можно использовать для создания временного псевдонима, который будет доступен только для компонентов доступа к данным внутри приложения. При щелчке на кнопке списка доступных псевдонимов свойства DatabaseName в Инспекторе объектов для любого компонента доступа к данным в списке будет доступен и временный псевдоним компонента TDatabase.

Например, при переключении приложения на другую базу данных можно изменить только значение псевдонима в компоненте TDatabase. Если все компоненты наборов данных подключены к временному псевдониму компонента TDatabase, то они автоматически переключатся на новую БД.

Дополнительные возможности управления наборами данных при переключении соединения предоставляют свойства Connected и KeepConnection. Они позволяют одновременно с соединением закрыть все активные наборы данных.

Если наборы данных приложения подключены к базе данных через компонент TDatabase, то перед их открытием необходимо установить соединение с БД. Соединение с БД устанавливается при помощи метода open. Если попытаться активизировать набор данных без этого метода, то соединение будет установлено автоматически.

Аналогичная картина возникает при закрытии наборов данных и отключении от БД. Дополнительное средство управления в этом случае предоставляет свойство KeepConnection. Если оно равно значению True, то при закрытии последнего открытого набора данных соединение остается открытым. В противном случае соединение автоматически закрывается.

Это позволяет управлять соединением в различных исходных ситуациях. При большой загруженности сервера бывает необходимо прерывать соединение каждый раз. Если требуется разгрузить сетевой график, то соединение лучше оставлять включенным.

При подключении к базе данных довольно часто требуется задать значения для параметров драйвера BDE. Для этого используется свойство Params, представляющее собой обычный список. В нем необходимо задавать названия изменяемых параметров и их новые значения:

```
USERNAME=SYSDBA
PASSWORD=masterkey
```

Значения параметров можно задавать как статически, так и динамически во время выполнения.

Компонент TDatabase может облегчить подключение к базам данных с регистрацией пользователей. При регистрации на сервере достаточно задать имя пользователя, пароль в свойстве Params (см. выше) и установить для свойства LoginPrompt значение False. Эта комбинация работает как во время выполнения, так и во время разработки.

**Примечание**

*Для организации доступа к защищенным паролем таблицам Paradox используется метод AddPassword компонента TSession (см. выше).*

Дополнительные возможности обработки регистрации пользователя дает единственный метод-обработчик onLogin, программный код которого выполняется вместо появления стандартного диалога ввода имени и пароля. Это позволяет разработчику создавать собственные сценарии регистрации пользователей.

Для обеспечения доступа к функциям API BDE используется свойство Handle (BDE играет важную роль при создании соединения).

Управление выполнением транзакций осуществляется при помощи методов StartTransaction, Commit и RollBack.

Компоненты доступа к данным

Компоненты доступа к данным, используемые при разработке приложений BDE, располагаются на странице BDE Палитры компонентов. Их общими предками являются классы TBDEDataSet и TDBDataSet (см. рис. 12.1). Они обеспечивают работоспособность основных компонентов доступа к данным BDE — TTable, TQuery, TStoredProc.

**Класс TBDEDataSet**

Этот класс является потомком класса TDataSet, его значение трудно переоценить: именно TBDEDataSet обеспечивает работоспособность важнейших механизмов набора данных за счет обращения к функциям BDE (табл. 16.6). Например, класс TBDEDataSet перекрывает абстрактные методы своего предка TDataSet, отвечающие за такие важнейшие операции, как чтение данных и сохранение изменений в базе данных, навигация по записям набора данных, фильтрация.

Напомним, что все эти механизмы не созданы "с нуля", а только дополнены обращениями к функциям BDE в необходимых местах методов, изначально описанных в классе TDataSet. Например, для обеспечения фильтрации записей набора данных к классу добавлено новое свойство:

```
type
TFilterOption = (foCaseInsensitive, foNoPartialCompare);
TFilterOptions = set of TFilterOption;
property FilterOptions: TFilterOptions;
```

Оно определяет дополнительные параметры отбора записей по фильтру (чувствительность к регистру символов и отбор по текстовому шаблону).

Дополнительно к существующим добавлен механизм кэширования изменений. Теперь все вносимые пользователем изменения могут накапливаться в специальном буфере, а их передачей в базу данных можно управлять.

**Примечание**

*Эта возможность очень полезна при создании клиентских приложений в архитектуре клиент/сервер и играет ключевую роль при обеспечении возможности редактирования наборов данных сложных запросов SQL.*

Дополнительно к методам работы с полями класса TDataSet добавлены функции использования полей в формате BLOB.

Для обеспечения использования функций API BDE на программном уровне добавлено свойство, содержащее дескриптор курсора, соответствующего текущей записи набора данных:

```
type HDBICur: Longint;
property Handle: HDBICur;
```

Также класс обеспечивает возможность программного управления вторичными индексами набора данных в зависимости от типа таблицы базы данных.

Таблица 6.6. Свойства и методы класса TBDEDataSet

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property BlockReadSize: Integer; | Pu | Определяет размер буфера при блочном чтении данных. Такой режим используется для быстрого перемещения по большим массивам данных. Если значение свойства больше нуля, навигация по набору данных осуществляется без изменения состояния компонентов отображения данных и вызова методов-обработчиков событий |
| property CacheBlobs: Boolean; | Pu | Разрешает использование буфера памяти для данных типа BLOB |
| property CachedUpdates: Boolean; | Pb | Включает или отключает режим кэширования изменений в наборе данных. Используется в клиентских приложениях архитектуры клиент/сервер |
| property CanModify: Boolean; | Pu, Ro | Если набор данных позволяет делать изменения, свойство возвращает True, иначе — False |
| property Explndex: Boolean; | Pu, Ro | Показывает, используются ли в наборе данных индексы dBASE |
| property Filter: string; | Pb, Ro | Содержит выражение для фильтра набора данных |
| property Filtered: Boolean; | Pb, Ro | Управляет включением фильтра набора данных |

| | | |
|---|---|---|
| TFilterOption = (foCaselnsensitive, foNoPartialCompare) ; property FilterOptions: TFilterOptions; | Pb | Определяет параметры фильтра:<br>• foCaselnsensitive — строковые значения фильтруются без учета регистра;<br>• foNoPartialCompare — при фильтрации символ "*" рассматривается как обычный символ, иначе он означает, что на этом месте может находиться произвольное подмножество любых символов |
| type HDBICur: Longint; property Handle: HDBICur; | Pu, Ro | Указатель на курсор BOE, связанный с текущей записью набора данных |
| property KeySize: Word; | Pu, Ro | Содержит размер ключа для текущего индекса набора данных |
| type TLocale: Pointer; property Locale: TLocale; | Pu, Ro | Указатель на языковый драйвер BOE |
| property RecNo: Longint; | Pu | Номер текущей записи набора данных |
| property RecordCount: Longint; | Ro | Содержит число записей в наборе данных |
| property RecordSize: Word; | Ro | Содержит размер одной записи набора данных |
| property UpdateObject: TDataSetUpdateObject; | Pu | Экземпляр объекта TUpdateObject, используемого при кэшировании изменений |
| type TUpdateRecordTypes = set of (rtModified, rtlnserted, rtDeleted, rtUnmodified) ; property UpdateRecordTypes : TUpdateRecordTypes ; | PU | Определяет видимость записей в режиме кэширования изменений в зависимости от их состояния:<br>• rtModified — доступны измененные записи;<br>• rtinserted — доступны добавленные записи;<br>• rtDeleted — доступны удаленные записи;<br>• rtUnmodified — доступны немодифицированные записи |
| property UpdatesPending: Boolean; | Ro | Значение True говорит о том, что буфер изменений при кэшировании содержит не сохраненные на сервере изменения |
| **Методы** | | |
| procedure ApplyUpdates; | Pu | Записывает изменения из буфера в базу данных в режиме кэширования |

| | | |
|---|---|---|
| function BookmarkValid (Bookmark : TBookmark): Boolean; overrider | Pu | Проверяет существование экземпляра закладки, передаваемого в параметре Bookmark |
| procedure Cancel; | Pu | Отменяет все изменения, сделанные в текущей записи с момента последнего сохранения |
| procedure CancelUpdates; | Pu | Отменяет все изменения, сделанные с момента последней записи в базу данных и очищает буфер в режиме кэширования |
| procedure CommitUpdates; | Pu | Очищает буфер изменений в режиме кэширования |
| function CompareBookmarks (Bookmarkl, Bookmark2: TBookmark): Integer; | Pu | Проверяет идентичность закладок, указанных в параметрах Bookmark1 и Bookmark2. При значении сигнализирует о наличии отличий в двух закладках |
| function ConstraintCallBack (Req: DsInfoReq; var ADataSources : DataSources) : DBIResult; stdcall; | Pu | Обеспечивает доступ к функциям ограничения данных API BDE |
| function ConstraintsDisabled: Boolean; | Pu | Показывает, включены или отключены ограничения данных |
| function CreateBlobStream ( Field : TField; Mode: TblobStreamMode) : TStream; override; | Pu | Создает поток для чтения/записи данных типа BLOB |
| procedure DisableConstraints; | Pu | Отключает ограничения данных |
| procedure EnableConstraints; | Pu | Включает ограничения данных |
| procedure FetchAll; | Pu | Переносит все изменения из буфера и восстанавливает все записи от текущей позиции до конца набора данных |
| procedure FlushBuffers; | Pu | Передает в базу данных все изменения из буфера записи |

| | | |
|---|---|---|
| function GetBlobFieldData (FieldNo: Integer; var Buffer: TblobByteData): Integer; override; | Pu | Читает все данные BLOB из поля FieldNo В буфер Buffer |
| function GetCurrentRecord (Buffer : PChar) : Boolean | Pu | Помещает текущую строку в буфер Buffer |
| procedure Getlndexlnfo; | Pu | Обновляет информацию о текущем индексе набора данных |
| function IsSequenced: Boolean; override; | Pu | Определяет, поддерживает ли таблица БД нумерацию последовательности записей. В классе TDataSet всегда возвращает True, т. к. абстрактный набор данных свободен от конкретной реализации БД и всегда нумерует записи |
| function Locate (const KeyFields: string; const KeyValues : Variant; Options: TlocateOptions) : Boolean; | Pu | Осуществляет поиск в наборе данных. Параметр KeyFields содержит список полей, по которым ведется поиск. Параметр KeyValues содержит значения полей для поиска. Параметр Options определяет условия поиска. Если запись найдена, курсор набора данных устанавливается на эту запись и возвращается True |
| function Lookup (const KeyFields: string; const KeyValues : Variant; const ResultFields: string): Variant; | Pu | Осуществляет поиск в наборе данных. Возвращает массив значений требуемых полей найденной записи. Параметры аналогичны методу Locate |
| procedure Post; override; | Pu | Пересылает сделанные в текущей записи изменения в базу данных |
| procedure RevertRecord; | Pu | Отменяет все изменения в текущей строке при работающем буфере изменений |
| procedure Translate (Src, Dest : PChar; ToOem: Boolean); override; | Pu | Форматирует текст. Если параметр ToOem = True, текст Src в формате ANSI переводится в текст Dest в формате OEM и наоборот |
| function UpdateStatus: TUpdateStatus; | | Возвращает тип сохраняемых в буфере изменений данных (см. табл. 6.1) |
| **Методы-обработчики событий** | | |

| | | |
|---|---|---|
| TUpdateAction = (uaFail, uaAbort, uaSkip, uaRetry, uaApplied); TUpdateErrorEvent = procedure (DataSet: TDataSet; E: EDatabaseError; UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction) of object; property OnUpdateError: TUpdateErrorEvent; | Pu | Вызывается при возникновении ошибки переноса кэшированных в буфере изменений в таблицу базы данных |
| type TUpdateAction = (uaFail, uaAbort, uaSkip, uaRetry, uaApplied) ; TUpdateRecordEvent = procedure ( DataSet : TDataSet ; UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction) of object; property OnUpdateRecord : TUpdateRecordEvent ; | Pu | Вызывается при сохранении кэшированных в буфере изменений для отдельной записи. Применяется для организации дополнительного управления этим процессом, например для контроля какого-либо конкретного значения |

**Класс TDBDataSet**

Класс TDBDataSet является непосредственным предком основных компонентов доступа к данным ttable, TQuery и TstoredProc. Новые свойства и методы класса обеспечивают соединение набора данных с базой данных и используют функции BDE (табл. 6.7).

В процессе соединения важнейшую роль играет свойство DatabaseName, которое должно содержать псевдоним или полный путь к файлам БД. Для управления отдельным соединением с базой данных можно применять специальный компонент TDatabase. Указатель на экземпляр такого компонента содержится в свойстве Database.

Многие функции API BDE используют в своей работе дескриптор специальной структуры, описывающей подключенную базу данных. Доступ к этому дескриптору можно получить через свойство DBHandie.

Приложение баз данных одновременно может использовать несколько наборов данных, каждый из которых подключен к собственной базе данных. Совокупность соединений управляется в рамках сеанса работы, который инкапсулируется компонентом TSession. Указатель на экземпляр такого

компонента можно использовать в наборе данных при помощи свойства DBSession.

Для работы с удаленными серверами в класс введено свойство Provider, обеспечивающее доступ к интерфейсу iProvider.

Таблица 16.7. Свойства и методы класса TDBDataSet

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property AutoRefresh: Boolean; | Pb | При значении True все автоматически создаваемые значения полей (автоинкрементные, значения по умолчанию) обновляются автоматически |
| property Database: TDatabase; | Pu, Ro | Указатель связанного с набором данных компонента TDatabase |
| property DatabaseName: string; | Pu, Pb | Псевдоним базы данных |
| type HDBISES: Longint; property DBHandie: HDBISES; | Pu, Ro | Дескриптор базы данных. Используется при работе с API BDE |
| type TLocale: Pointer; property DBLocale: TLocale; | Pu, Ro | Идентифицирует языковый драйвер API BDE |
| property DBSession: TSession | Pu, Ro | Указатель для компонента TSession, с которым работает набор данных |
| property Provider: IProvider; | Pu, Ro | Идентифицирует интерфейс IProvider |
| property SessionName: string; | Pu, Ro | Содержит имя компонента сеанса, в котором работает набор данных |
| **Методы** | | |
| function CheckOpent Status: DBIResult): Boolean; | Pu | Возвращает результат вызова BDE. Используется для тестирования соединения |
| procedure CloseDatabase ( Database: TDatabase); | Pu | Закрывает связь с базой данных, определяемой параметром Database |
| procedure GetProviderAttributes (List: TList); override; | Pu | Возвращает в списке List параметры языкового драйвера |
| function OpenDatabase: TDatabase; | Pu | Открывает связь с базой данных, определяемой свойством DatabaseName |

**Компонент TTable**

Компонент Ttable инкапсулирует таблицу реляционной базы данных, причем независимо от типа базы данных. Для доступа к данным компонент использует функции BDE (см. выше).

Необходимая для работы база данных задается свойством DatabaseName, в котором можно указать зарегистрированный в BDE псевдоним БД или полный путь к файлам БД.

Таблица БД, на основе которой создается набор данных, определяется свойством TableName. При необходимости тип таблицы задается свойством TableType, хотя обычно это свойство имеет значение ttDefault (см. табл. 6.4), которое включает автоматическое определение типа таблицы по расширению файла.

**Примечание**

*Свойство TableType работает только в локальных БД. Обратите внимание, что возможные значения свойства соответствуют основным типам локальных драйверов BDE.*

При помощи методов Open и close набор данных открывается и закрывается. О его состоянии можно судить по значению свойства Active. Более подробно о состоянии набора данных расскажет свойство state (см. ниже).

Записи в набор данных можно отбирать при помощи свойств Filter, Filtered, FilterOptions, создающих фильтр, ограничивающий набор данных по значениям данных в одном или нескольких полях.

Методы SetRangeStart, SetRangeEnd, SetRange, ApplyRange, EditRangeStart, EditRangeEnd создают специальный диапазон включаемых в набор данных записей, отбор в диапазон проводится по задаваемым граничным значениям любых полей набора данных.

Поиск нужной записи можно осуществлять методами Lookup или Locate (достаточно просто, но не очень быстро) или, используя существующие в таблице базы данных индексы, методом FindKey (сложнее, но очень быстро).

От предков компонент унаследовал инструменты для работы с закладками. Это свойство Bookmark и методы GetBookmark, FreeBookmark, GotoBookmark.

Работа с полями осуществляется целой группой свойств и методов, среди которых особое место занимает свойство Fields, представляющее собой индексированный список всех полей набора данных. Это свойство удобно использовать в процессе разработки для организации доступа к полям.

Использование индексов обеспечено свойствами indexName, indexFieids, IndexFieldNames, IndexFiles.

Свойства MasterSource, MasterField, IndexName дают возможность установить отношение типа главный/подчиненный с другой таблицей.

Очень полезны в практическом использовании методы и свойства для работы с буфером изменений (свойства CachedUpdates, PendingUpdates, UpdateRecordTypes, МСТОДЫ ApplyUpdates, CancelUpdates, CommitUpdate, RevertRecord). Буфер применяется в клиентских приложениях многоуровневых систем доступа к данным.

От классов TDataSet и TBDEDataSet унаследован обширный набор методов-обработчиков событий, позволяющий решать любые задачи по управлению набором данных.

В табл. 6.8 приведена справочная информация о свойствах и методах компонента ttable. После этого рассматриваются подробности применения основных механизмов набора данных.

Таблица 16.8. Свойства и методы класса Ttablel

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property DataSource: TDataSource; | Pu, Ro | Ссылается на компонент TDataSource главного набора данных в отношении главный/подчиненный |
| property Defaultlndex: Boolean; | Pb | Управляет сортировкой данных. При значении True записи упорядочиваются по первичному ключу. При значении False упорядочивание не производится |
| property Exclusive: Boolean; | Pb | Ограничивает доступ к таблице. При значении True с таблицей может работать только одно приложение. Это свойство важно при одновременной работе нескольких приложений с данными в локальной сети |
| property Exists: Boolean; | Pu, Ro | Значение True говорит о том, что связанная с компонентом таблица базы данных существует |
| property IndexDef s: TindexDefs; | Pb | Содержит информацию об индексах таблицы |
| property IndexFieldCount: Integer; | Pu, Ro | Возвращает число полей в текущем индексе таблицы |
| property IndexFieldNames: string; | Pb | Разделенный запятыми список названий полей, составляющих текущий индекс. Используется для таблиц серверов SQL |
| property IndexFields: [Index: Integer] : TField; | Pu | Индексированный список полей текущего индекса |
| property IndexFiles: TStrings; | Pb | Список индексных файлов для таблиц dBASE |
| property IndexName: string; | Pb | Определяет вторичный индекс для таблицы. Используется для таблиц локальных СУБД |

| | | |
|---|---|---|
| property KeyExclusive: Boolean; | Pu | Управляет границами диапазона, задаваемого методом SetRange. При значении True крайние записи в диапазон не включаются |
| property KeyFieldCount: Integer; | Pu | Содержит число полей ключа, используемых при поиске. При значении 0 применяется только первое поле, при значении 1 — два первых поля и т. д. По умолчанию устанавливается полное число полей ключа |
| property MasterFields: string; | Pb | Список имен полей главной таблицы, разделенных запятой, используемых при создании отношения главный/подчиненный |
| property MasterSource: TDataSource; | Pb | Содержит имя компонента TDataSource, связанного с набором данных, который является главным в отношении главный/подчиненный |
| property Readonly: Boolean; | Pb | Включает и отключает режим "только для чтения". В некоторых случаях набор данных можно открыть только в этом режиме |
| property StoreDef s: Boolean; | Pb | При значении True все сведения об индексах и структуре таблицы хранятся вместе с формой или модулем данных. В этом случае при создании набора данных одновременно создаются поля, индексы, ограничения |
| property TableLevel: Integer; | Pu | Содержит значение уровня таблицы, используемого в драйвере BOE |
| property TableName: TFileName; | Pb | Определяет имя таблицы |
| type TTableType = (ttDefault, ttParadox, ttDBase, ttASCII, ttFoxPro) ; property TableType: TTableType; | Pb | Определяет тип таблицы для стандартного драйвера BOE. Значение ttDefault означает, что тип таблицы определяется по расширению файла |
| **Методы** | | |
| procedure Addlndex (const Name, Fields: string; Options: TIndexOptions) ; | Pu | Создает новый индекс. Параметр Name определяет имя нового индекса, параметр Fields — список полей индекса через запятую, параметр Options задает тип индекса |

| | | |
|---|---|---|
| procedure ApplyRange; | Pu | Включает в работу границы диапазона, заданные методами SetRangeStart, SetRangeEnd или EditRangeStart, EditRangeEnd |
| type TBatchMode = (batAppend, batUpdate, batAppendUpdate , batDelete, batCopy) ; function BatchMove (ASource: TBDEDataSet; AMode : TBatchMode) : Longint; | Pu | Переносит записи из таблицы ASource в набор данных. Тип операции задается параметром AMode. Возвращает число обработанных записей |
| procedure CancelRange; procedure CloselndexFile (const IndexFileName: string) ; | Pu Pu | Удаляет текущий диапазон Закрывает индексный файл для таблиц dBASE |
| procedure CreateTable; | Pu | Создает новую таблицу, основываясь на данных о структуре таблицы, содержащихся в свойствах FieldDef s и indexDef s. Если свойство FieldDef s пустое, используется свойство Fields. Структура и данные существующей таблицы перезаписываются |
| procedure Deletelndex (const Name: string); | Pu | Удаляет вторичный индекс |
| procedure DeleteTable; | Pu | Уничтожает таблицу базы данных. Набор данных при этом должен быть закрыт |
| procedure EditKey; | Pu | Переводит набор данных в режим редактирования буфера поиска. После использования этого метода можно изменять значения полей, которые применяются для поиска записей |
| procedure EditRangeEnd; | Pu | Разрешает редактирование нижней границы диапазона |
| procedure EditRangeStart; | Pu | Разрешает редактирование верхней границы диапазона |
| procedure EmptyTable; | Pu | Удаляет все записи из набора данных |

| | | |
|---|---|---|
| function FindKey (const KeyValues: array of const) : Boolean; | Pu | Проводит поиск записи, значения полей которой удовлетворяют условиям, заданным параметром KeyValues. Значения разделяются запятыми. Для поиска можно использовать только поля, входящие в текущий индекс. Для локальных стандартных таблиц BDE это поля, определяемые свойством indexName. Для таблиц серверов SQL индекс можно задать свойством indexFieldNames. При успешном поиске функция возвращает значение True |
| procedure FindNearest (const KeyValues: array of const); | Pu | Проводит поиск записи, значения полей которой, заданные параметром KeyValues, в минимальной степени отличаются от требуемых в большую сторону. Значения для поиска разделяются запятыми. Для поиска можно использовать толькополя, входящие в текущий индекс. Для локальных стандартных таблиц BOE это поля, определяемые свойствомIndexName. Для таблиц серверов SQL индекс можно задать свойством indexFieldNames. При успешном поиске функция возвращает True |
| procedure GetlndexNames (List : TStrings) ; | Pu | Возвращает список индексов таблицы |
| procedure GotoCurrent (Table: TTable) ; | Pu | Синхронизирует курсор набора данных с курсором таблицы, заданной параметром Table |
| function GotoKey: Boolean; | Pu | Устанавливает курсор на запись, соответствующую значениям полей, заданным при последнем применении метода SetKey или EditKey |
| procedure GotoNearest; | Pu | Устанавливает курсор на запись, точно соответствующую значениям полей, заданным при последнем применении метода SetKey или EditKey, или следующую ближайшую к ним по значениям |
| type TLockType = (ItReadLock, ItWriteLock) ; procedure LockTable ( LockType : TLockType) ; | Pu | Закрывает доступ к таблице Paradox или dBASE из других приложений |

| | | |
|---|---|---|
| procedure OpenlndexFile (const IndexFileName: string); | Pu | 1 Открывает индексный файл таблицы dBASE |
| procedure RenameTable (const NewTableName: string); | Pu | Переименовывает таблицу Paradox или dBASE |
| procedure SetKey; | Pu | Очищает буфер поиска. После использования этого метода можно изменять значения полей, используемые для поиска записей |
| procedure SetRange (const StartValues, EndValues: array of const) ; | Pu | Задает диапазон отбора записей. Параметр StartValues определяет значения полей для верхней границы диапазона. Параметр EndValues определяет значения полей для нижней границы диапазона. Значения диапазона задаются для полей текущего индекса |
| procedure SetRangeEnd; | Pu | Задает нижнюю границу диапазона. После этого метода необходимо задать значения для полей текущего индекса, которые и будут нижней границей |
| procedure SetRangeStart; | Pu | Задает верхнюю границу диапазона. После этого метода необходимо задать значения для полей текущего индекса, которые и будут верхней границей |
| type TLockType = (ItReadLock, ItWriteLock) ; procedure UnlockTable (LockType : TLockType) ; | Pu | Разблокирует таблицу Paradox или dBASE для доступа из других приложений |

**Компонент TQuery**

Компонент TQuery реализует все основные функции стандартного компонента запроса, описанные выше. Прямым предком компонента является класс TDBDataSet.

Для подключения к базе данных используется свойство DatabaseName, в котором задается псевдоним BDE или путь к базе данных.

Текст запроса определяется свойством SQL, для задания которого применяется простой редактор, открывающийся при щелчке на кнопке свойства в Инспекторе объектов (рис. 6.6).

Для управления текстом запроса во время выполнения приложения можно использовать возможности класса TStrings.

Основные свойства и методы компонента TQuery представлены в табл. 6.9.

Таблица 6.9. Свойства и методы компонента TQuery

| Объявление | Тип | Описание |
|---|---|---|
| **Свойства** | | |
| property Constrained: Boolean; | Pb | При значении True запрещает внесение в набор данных таких значений, которые не соответствуют условиям отбора запроса. Применимо для локальных БД |
| property DataSource: TDataSource; | Pb | Ссылается на компонент TDataSource, из набора данных которого задаются значения параметров |
| property Local: Boolean; | Ro | Значение True означает, что запрос обращается к локальной таблице |
| property ParamCheck: Boolean; | Pb | При значении True параметры запроса обновляются при изменении свойства SQL во время выполнения |
| property ParamCount: Word; | Ro | Возвращает число параметров в запросе |
| property Params [Index : Word]TParams; | Pb | Индексированный список объектов TParams, каждый из которых соответствует одному параметру запроса |
| property Prepared: Boolean | Pu | Возвращает результат выполнения операции подготовки запроса к выполнению |
| property RequestLive: Boolean; | Pu | При значении False результат запроса нельзя редактировать, независимо от того, редактируемый результат или нет. При значении True результат запроса можно редактировать, но только если он "живой" |
| property RowsAffected: Integer; | Ro | Возвращает число модифицированных записей набора данных с момента последнего выполнения запроса |
| property SQL: TStrings; | Pb | Содержит текст запроса |
| property SQLBinary: PChar; | Pu | Внутреннее свойство для обеспечения работы с BOE |
| property StmtHandle: HDBIStmt; | Ro | Возвращает экземпляр объекта, соответствующего запросу в BDE. Используется при прямом вызове функций BDE |
| property Text: PChar; | Ro | Указатель на символьный массив, содержащий передаваемый в BDE текст запроса |

| | | |
|---|---|---|
| property UniDirectional: Boolean; | Pb | Определяет тип используемого курсора данных |
| **Методы** | | |
| procedure ExecSQL; | Pu | Выполняет запрос без открытия набора данных |
| procedure GetDetailLinkFields (MasterFields, DetailFields: TList) ; override; | Pu | Заполняет списки параметров метода экземплярами объектов полей двух таблиц запроса, находящихся в отношении "один-ко-многим" |
| function ParamByName (const Value: string) : TParam; | Pu | Возвращает ссылку на экземпляр объекта параметра с именем, переданным в параметре Value |
| procedure Prepare; | PU | Готовит запрос к выполнению |
| procedure UnPrepare; | Pu | Освобождает ресурсы, занятые при подготовке запроса к выполнению |

### Компонент TStoredProc

Компонент TStoredProc обеспечивает использование в приложениях BDE хранимых процедур. Прямым предком компонента является класс TDBDataSet. Поэтому результатом выполнения хранимой процедуры может быть не только одиночный результат, но и полноценный набор данных.

Основные функции компонента TStoredProc соответствуют возможностям стандартного компонента хранимой процедуры, описанного выше. Свойства и методы компонента TStoredProc представлены в табл. 6.10.

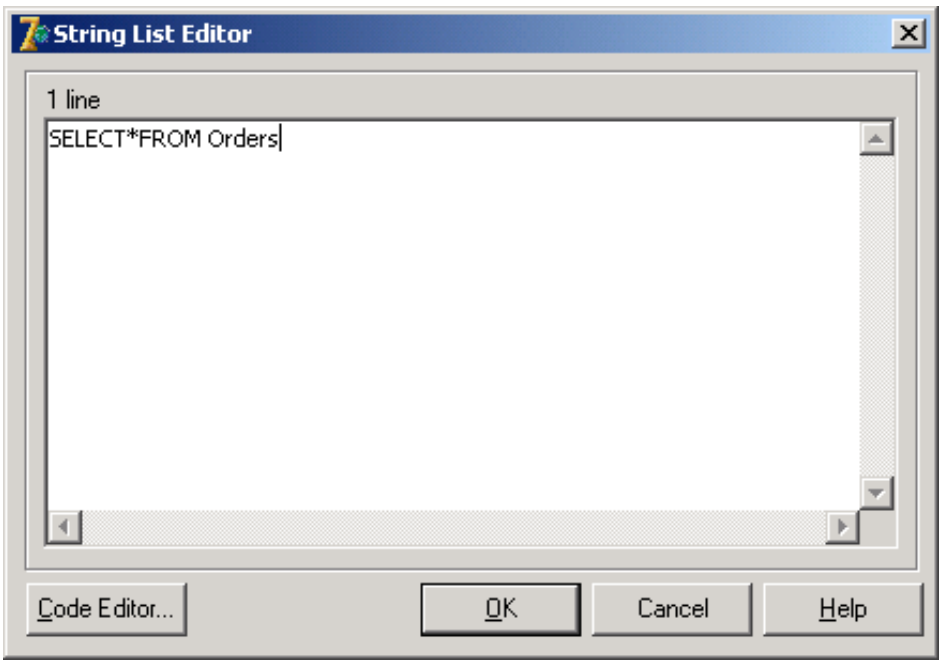


Рисунок 6.6. Редактор свойства SQL компонента TQuery

Средствами классов-предков выполняется и подключение компонента к базе данных. Свойство DatabaseName определяет базу данных.

Свойство storedProcName задает имя хранимой процедуры.

Перед выполнением хранимую процедуру необходимо подготовить. В частности, на этом этапе осуществляется передача параметров и выделение ресурсов. Эта операция выполняется автоматически при использовании методов ExecProc и Open, или задается явно методом Prepare.

Явная подготовка процедуры полезна при неоднократном вызове хранимой процедуры. Если перед первым вызовом процедуры выполнить метод Prepare, то все последующие вызовы будут осуществляться без подготовки, которая уже была сделана. В противном случае подготовка будет производиться автоматически перед каждым выполнением хранимой процедуры.

Таблица 6.10. Свойства и методы компонента TstoredProc

| **Объявление** | **Тип** | **Описание** |
|---|---|---|
| **Свойства** | | |
| property Overload: Word; | Pb | Идентификатор процедуры. Используется только для сервера Oracle |
| type TParamBindMode = (pbByName, pbByNumber) ; property ParamBindMode: TParamBindMode ; | Pb | Определяет порядок присваивания значений параметров:<br>• pbByName — по именам параметров;<br>• pbByNumber — по номерам параметров в списке свойства Params |
| property ParamCount: Word; | Ro | Возвращает общее число параметров |
| property Params: TParams; | Pb | Индексированный список параметров |
| property Prepared: Boolean; | Pu | Возвращает значение True, если подготовка процедуры уже проводилась |
| property StmtHandle: HDBIStmt; | Ro | Дескриптор выражения BOE. Используется при прямом вызове функций BDE |
| property StoredProcName: string; | Pb | Содержит имя хранимой процедуры |
| **Методы** | | |
| procedure CopyParams (Value : TParams) ; | Pu | Копирует параметры из списка Value |
| function DescriptionsAvailable : Boolean; | Pu | При значении True параметры хранимой процедуры доступны из приложения |
| procedure ExecProc; | Pu | Передает на сервер сигнал для запуска хранимой процедуры |
| procedure GetResults; | PU | Возвращает выходные параметры в приложение (используется только для сервера Sybase) |
| function Pa ramByName (const Value: string) : TParam; | Pu | Возвращает параметр с именем Value |
| procedure Prepare; | Pu | Готовит процедуру к выполнению |
| procedure UnPrepare; | Pu | Освобождает ресурсы, использованные во время подготовки процедуры |

**Форма контроля: тестовые вопросы**

1. Для чего предназначена команда GROUP BY предложения SELECT?
a) группировка данных полей
b) определение условий для вывода данных полей
c) получение итогов данных по столбцам
d) упорядочивание данных полей
e) объединение запросов

2. Для чего предназначена инструкция ORDER BY предложения SELECT?
a) упорядочивание данных полей
b) определение условий для вывода данных полей
c) группировка данных полей
d) получение итогов данных по столбцам
e) объединение запросов

3. Для чего используется предложение UNION?
a) для размещения нескольких запросов вместе и объединения их вывода
b) определение условий для вывода данных полей
c) группировка данных полей
d) получение итогов данных по столбцам
e) упорядочивание данных полей

4. Что будет получено в результате следующего запроса: SELECT * FROM STUDENTS?
a) вывод полного списка полей из таблицы STUDENTS
b) вывод первого поля из таблицы STUDENTS
c) вывод названий всех таблиц из базы данных STUDENTS
d) вывод всех не NULL – значений поля STUDENTS
e) создание таблицы STUDENTS

5. Как выделяется вложенный запрос в SQL – предложении?
a) берется в круглые скобки
b) берется в фигурные скобки
c) начинается специальным служебным словом
d) никак не выделяется, а просто следует во внутреннем предложении SELECT
e) нет правильного ответа

6. Что выполняет команда DELETE FROM <table name> WHERE <условие>?
a) удаляет данные из имеющейся таблицы базы данных
b) добавляет запись в имеющуюся таблицу базы данных
c) обновляет или изменяет данные в имеющейся таблице базы данных
d) создает новую таблицу в имеющейся базе данных
e) добавляет поля к вновь созданной таблице базы данных

7. Что выполняет оператор WHERE

a) определение условий для вывода данных полей

b) группировка данных полей

c) получение итогов данных по столбцам

d) упорядочивание данных полей

e) объединение запросов

8. Какой оператор языка SQL позволяет вводить пустое значение в поле вместо значения?

a) IS NULL

b) LIKE

c) BETWEEN

d) IN

e) WHERE

9. Для чего предназначена команда GROUP BY предложения SELECT?

a) группировка данных полей

b) определение условий для вывода данных полей

c) получение итогов данных по столбцам

d) упорядочивание данных полей

e) объединение запросов

10. Укажите правильную запись оператора, с которого начинаются все SQL – запросы:

a) SELECT.....FROM

b) CELECT.....FROM

c) SELEKT.....FROM

d) SELECT.....FOR

e) FROM.....SELECT

## Рекомендуемая литература

**Основная литература**

1. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.

2. Липаев В.В. Проектирование программных средств. -М.:Высшая школа,1990.

3. Майерс Г. Искусство тестирования программ.-М.: Финансы и статистика, 1982.

Фокс Дж. Программное обеспечение и его разработка.-М.: Мир, 1985.

4. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2007 г.

5. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.

**Дополнительная литература**

6. С.Бобровский. Delphi 7. Учебный курс. Издательство «ПИТЕР»,2005г.
7. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».

## Тема № 7
## Компоненты Rave Reports и отчеты в приложении Delphi

**План:**

1. Генератор отчетов Rave Reports 5.0
2. Компоненты Rave Reports и их назначение
3. Отчет в приложении Delphi
4. Компонент отчета TRvProject
5. Компонент управления отчетом TRvSystem

На первый взгляд кажется, что в сфере создания и печати отчетов в Delphi 7 произошла небольшая революция. Просматривая первый раз Палитру компонентов, вы не найдете в ней хорошо знакомой по прошлым версиям Delphi страницы **QReport.** Вместо старого генератора отчетов в состав Delphi 7 включен продукт Rave Reports 5.0 от фирмы Nevrona. "Ну и почему же это событие не дотягивает до революции в отчетах?" — спросит читатель. Авторы могут обосновать свою точку зрения.

Во-первых, компоненты QReport по-прежнему доступны разработчику — пакет DCLQRT70.BPL все так же занимает прочное место в папке \Delphi7\Bin и может быть установлен в Палитру компонентов обычным способом. Да и было бы странно ожидать другого от фирмы Borland, которая бдительно следит за обратной совместимостью приложений. Посмотрите к примеру на страницу **Win 3.1** Палитры компонентов — новые поколения программистов никогда не видели "прабабушку" Windows XP, и все же исторические компоненты занимают свое исконное место!

Во-вторых, схема создания и внедрения отчетов в приложения Delphi практически не изменилась. В Rave Reports имеются и глобальный класс отчета, и классы полос, и компоненты преобразования данных. Существенным нововведением можно считать только визуальную среду создания отчетов, что несомненно облегчит жизнь создателей отчетов и сделает их работу эффективнее и приятнее.

Тем не менее, в Delphi 7 генератор отчетов Rave Reports является основным средством создания отчетов и его компоненты устанавливаются в Палитре компонентов по умолчанию на странице **Rave.** Поэтому главы этой части посвящены разнообразным аспектам разработки отчетов в Rave Reports.

**Генератор отчетов Rave Reports 5.0**

Генератор отчетов Rave Reports 5.0 разработан фирмой Nevrona и входит в состав Delphi 7 в качестве основного средства для создания отчетов. Он состоит из трех частей:

- ядро генератора отчетов обеспечивает управление отчетом и его предварительный просмотр, и отправку на печать. Исполняемый код ядра сервера включается в приложение Delphi, делая его полностью автономным при работе с отчетами на компьютере клиента;
- визуальная среда разработки отчетов Rave Reports предназначена для разработки самих отчетов. Она позволяет добавлять к отчету страницы, размешать на них графические и текстовые элементы управления, подключать к отчетам источники данных и т. д. Отчеты сохраняются в файлах с расширением rav и должны распространяться совместно с приложениями, использующими их;
- компоненты Rave Reports расположены на странице **Rave** Палитры компонентов Delphi. Они обеспечивают управление отчетами в приложении.

Генератор отчетов устанавливается при инсталляции Delphi в папку \Delphi7\Rave5. Исходные коды компонентов разработчикам в Delphi недоступны.

Безусловно, визуальная среда разработки заметно упрощает процесс создания отчетов и позволяет добиться лучших результатов меньшими усилиями, чем в генераторе отчетов Quick Report, который использовался в предыдущих версиях Delphi. Тем не менее, при первом знакомстве с продуктом заметны и его недостатки. Система помощи оставляет тягостное впечатление не только своей крайней лаконичностью, но и фактическими ошибками. Многие свойства и методы остались недокументированными, и наоборот — имеющиеся в статьях подсказки описания не имеют реальных аналогов в коде компонентов.

Однако будем надеяться, что недостатки будут со временем исправлены. А мы займемся детальным знакомством с процессом создания отчетов.

Компоненты Rave Reports и их назначение

Компоненты для создания отчетов и управления расположены на странице **Rave** Палитры компонентов. Они делятся на следующие функциональные группы.

- Компонент отчета TRvproject, с точки зрения приложения, и есть отчет. Он обеспечивает загрузку заранее созданного в визуальной среде Rave Reports отчета из файла с расширением rav.

Подробнее об использовании компонента TRvproject рассказывается в ниже в этой главе.

- Компонент управления отчетом TRvSystem обеспечивает работу приложения с отчетом. Взаимодействуя с компонентом отчета, с одной стороны, и сервером отчета Rave Reports, с другой, этот компонент обеспечивает просмотр и печать отчетов.

Подробнее об использовании компонента TRvSystem рассказывается в ниже в этой главе.

- Компоненты соединения с источниками данных предназначены для подключения различных источников данных к отчетам. При этом могут использоваться технологии доступа к данным ADO, BDE, dbExpress.

К этой группе относятся компоненты:

- TRvCustomConnection;
- TRvDataSetConnection;
- TRvTableConnection;
- TRvQueryConnection.
- Компоненты преобразования данных позволяют конвертировать отчеты из формата данных Rave Reports в другие форматы (текстовый, PDF, HTML, RTF), а также распечатывать или просматривать отчеты.

К этой группе относятся компоненты:

- TRvNDRWriter;
- TRvRenderHTML;
- TRvRenderPreview;
- TRvRenderRTF;
- TRvRenderPrinter;
- TRvRenderText.
- TRvRenderPDF;

Подробнее об использовании компонентов преобразования данных рассказывается ниже в этой главе.

Отчет в приложении Delphi

Завершив обзор нового генератора отчетов, давайте обратимся к деталям программирования и посмотрим, что нужно сделать, чтобы приложение могло работать с отчетами.

Основой отчета является файл отчета с расширением rav. Он создается в визуальной среде разработки Rave Reports и может содержать произвольное число страниц. Каждая страница может быть оформлена графическими или текстовыми элементами или отображать данные из какой-либо базы данных. Другими словами, файл RAV — это проект будущего отчета, содержащий общую информацию об отчете, оформление его страниц и правила их заполнения.

После создания проект отчета необходимо связать с приложением Delphi. Для этого используется компонент TRvProject. Этот компонент обеспечивает представление отчета в приложении.

Но этого недостаточно, чтобы просмотреть или напечатать отчет. Для выполнения этих операций используется код ядра генератора отчета, который автоматически прикомпилируется к исполняемому коду приложения при переносе на любую форму проекта компонентов TRvProject и TRvSystem. Для управления операциями печати и просмотра в проекте должен присутствовать компонент TRvSystem.

Для того чтобы приложение Delphi могло выполнять функции печати отчетов, разработчик должен выполнить следующий набор операций.

1. При помощи визуальной среды разработки Rave Reports необходимо создать проект отчета и сохранить его.

2. Перенести в проект приложения в Delphi компонент TRvProject и связать его с файлом проекта отчета (см. ниже) при помощи свойства ProjectFile.

3. Перенести в проект приложения в Delphi компонент TRvSystem и связать его с компонентом TRvProject. ля этого используется свойство Engine компонента TRvProject (см. ниже).

4. Написать код приложения, обеспечивающий просмотр и печать отчета (при необходимости и другие операции), используя методы компонента TRvProject (см. ниже).

Конечно же, это наиболее простой способ включения отчета в приложения. Для решения более сложных задач необходимо изучить использованные выше компоненты более детально.

Компонент отчета TRvProject

Компонент TRvProject обеспечивает представление в приложении отчета. Для того чтобы связать проект отчета Rave Reports с компонентом, используется свойство

property ProjectFile: string;

До начала печати необходимо связать компонент TRvProject с компонентом управления отчетом TRvSystem. Для этого достаточно передать в свойстве

property Engine: TRpComponent;

ссылку на компонент TRvSystem.

При необходимости вы можете загрузить отчет из внешнего файла или потока:

procedure LoadFromFile(FileName: String);

procedure LoadFromStream(Stream: TStream);

Загруженный отчет становится текущим.

Кроме этого существует и пара методов для сохранения отчета:

function SaveToFile(FileName: String);

procedure SaveToStream(Stream: TStream);

В процессе работы приложения может потребоваться напечатать несколько различных отчетов. Для этого можно использовать требуемое число компонентов TRvProject или загружать нужные отчеты по мере необходимости.

Забегая немного вперед (см. гл. 24), скажем, что один файл проекта отчета может содержать несколько независимых отчетов. Каждый из них идентифицируется в компоненте TRvProject тремя свойствами. Имя, полное имя и описание отчета содержатся соответственно в трех свойствах только для чтения:

property ReportName: String;

property ReportFullName: String;

property ReportDesc: String;

При этом эти три свойства возвращают параметры текущего отчета. Сразу после загрузки из файла текущим становится отчет, являющийся отчетом по умолчанию в среде разработки. При необходимости сменить текущий отчет используется метод

function SelectReport(ReportName: string; FullName: boolean): boolean;

В параметре ReportName передается имя нужного отчета. Если параметр FullName имеет значение True, то это полное имя отчета, иначе — имя отчета.

В случае, если проект содержит несколько отчетов, их имена доступны при помощи метода

procedure GetReportList(ReportList: TStrings;FullName: boolean);

Список имен будет возвращен в список строк ReportList, а параметр FullName определяет, какие именно имена будут занесены в список. При значении параметра True метод возвращает полные имена отчетов (соответствует свойству ReportFullName), иначе — имена (соответствует свойству ReportlName).

Например, код

```
var ReportList: TStringList; i: Integer;
ReportList := TStringList.Create;
RvProjectl.Open;
try
RvProjectl.GetReportList(ReportList, False);
for i := 0 to ReportList.Count - 1
do RvProjectl.ExecuteReport(ReportList[I]);
finally
RvProjectl.Close;
ReportList.Free;
end;
```

последовательно печатает все отчеты, входящие в состав файла проекта отчета.

Файл проекта отчета можно включить в состав исполняемого файла приложения. Для этого используется свойство

property StoreRAV: Boolean;

При щелчке на кнопке в строке этого свойства в Инспекторе объектов открывается специализированный редактор **Load Into exe.**

Здесь можно задать файл проекта отчета. После этого в Инспекторе объектов в строке свойства storeRAv появятся дата и время загрузки проекта отчета. Это же время и дата будут сохранены в свойстве

property RaveBlobDateTime: TDateTime;

Отправить отчет на печать можно методом

procedure Execute;

или же методом

procedure ExecuteReport(ReportName: string);

который позволяет направить на печать отчет, заданный параметром ReportName. Он должен соответствовать имени отчета, хранящемуся в свойстве ReportName компонента TRvProject.

Отчет, содержащийся в компоненте Trvproject, может быть открыт для редактирования методом

procedure Open;

Не открывая отчет, вы не сможете использовать большинство свойств и методов компонента. Дело в том, что при открытии компонент загружает отчет из файла проекта или прикомпилированного кода (в случае использования свойства StoreRAV).

Сохранение и закрытие отчета соответственно выполняются методами

procedure Save; procedure Close;

Кроме этого, действия, аналогичные методам open и close, выполняются свойством

property Active: Boolean;

Если свойству присвоить значение True — отчет открывается, иначе — закрывается.

До и после открытия и закрытия отчета вызывается четверка методов-обработчиков:

property aeforeOpen: TNotifyEvent;

property AfterOpen: TNotifyEvent;

property BeforeClose: TNotifyEvent;

property AfterClose: TNotifyEvent;

Компонент управления отчетом TRvSystem

Компонент управления отчетом TRvSystem обеспечивает выполнение основных операций с отчетом из приложения. В приложении он должен быть связан с компонентом TRvProject (см. выше разд. "Компонент отчета TRvProject" данной главы). Этого вполне достаточно, чтобы компонент TRvSystem выполнил свою работу. У разработчика нет необходимости вызывать какие-либо методы компонента, чтобы направить отчет на печать.

В его составе инкапсулированы объекты, обеспечивающие вывод отчета из компонента TRvProject в один из трех системных приемников:

- файл (объект класса TSystemFiler);
- предварительный просмотр (объект класса TSystemPreview);
- принтер (объект класса TSystemPrinter).

За это отвечает свойство

type

TReportDest = (rdPreview, rdPrinter, rdFile);

property ReportDest: TReportDest;

которое может принимать одно из трех значений типа TReportDest.

Соответственно, для каждого типа системного приемника имеется свойство, позволяющее задать все его основные параметры.

Для вывода в файл это комплексное свойство

property SystemFiler: TSystemFiler;

Внутри него задается имя файла во вложенном свойстве

property FileName: string;

но при этом вложенное свойство

type

TStreamMode = (smMemory, smTempFile, smFile, sraUser); property StreamMode: TStreamMode;

должно иметь значение smFile.

При выводе отчета для предварительного просмотра используется экземпляр класса TSystemPreview, который доступен через свойство

property SystemPreview: TSystemPreview;

Его свойства совпадают со свойствами компонента TRvRenderPreview.

Перед открытием окна предварительного просмотра вызывается метод-обработчик

property OnPreviewShow: TNotifyEvent;

За вывод отчета на печать отвечает инкапсулированный в компоненте объект типа TSystemPrinter. К нему можно обратиться при помощи свойства

property SystemPrinter: TSystemPrinter;

Его свойства совпадают со свойствами компонента TRvRenderPrinter.

Перед тем как отправить отчет одному из трех системных приемников, компонент открывает диалог настройки печати.

property TitleSetup: TFormatString;

Перед открытием этого окна вызывается метод-обработчик

property OnPreviewSetup: TNotifyEvent;

Кроме этого, для диалога настройки печати можно задать ряд дополнительных параметров. Это делается в свойстве

type

TSystemSetup = (ssAllowSetup, ssAllowCopies, ssAllowCollate, ssAllowDuplex, ssAllowDestPreview, ssAllowDestPrinter, ssAllowDestFile, ssAllowPrinterSetup);

TSystemSetups = set of TSystemSetup;

property SystemSetups: TSystemSetups;

Элементы множества TSystemSetup означают следующее:

- ssAllowSetup — разрешает или запрещает использование диалога настройки печати компонента;
- ssAllowCopies — управляет доступностью установки числа копий отчета;
- ssAllowCollate — разрешает или запрещает настройку режима печати с разбором страниц по копиям;
- ssAllowDuplex — разрешает или запрещает настройку двусторонней печати;
- ssAllowDestPreview — разрешает или запрещает использование окна предварительного просмотра;
- ssAiiowDestPrinter — разрешает или запрещает использование принтера;
- ssAllowDestFile — разрешает или запрещает использование файла для вывода отчета;
- ssAiiowPrinterSetup — разрешает или запрещает использование диалога настройки параметров принтера.

Во время выполнения любой из перечисленных операций вывода отчета открывается окно состояния процесса. Его заголовок определяется свойством

property TitleStatus: TFormatString;

В нем отображается информационная строка состояния, которая может быть настроена при помощи свойств объекта SystemFiler, представленного в компоненте TRvSystem одноименным свойством.

Вложенное свойство

property StatusFormat: string;

определяет строку форматирования для текста о состоянии процесса. Для нее предусмотрены следующие управляющие символы:

- %c — текущее состояние процесса вывода;
- %p — номер текущей страницы;

- %f — номер первой страницы;
- %1 — номер последней страницы;
- %d — название устройства вывода (название принтера, имя файла, предварительный просмотр);
- %r — имя драйвера устройства вывода;
- %s — общее число страниц;
- %t — порт печати;
- %o-%9 — номера строк для свойства statusText (см. ниже).

Вложенное свойство

property StatusText: TStrings;

позволяет задать до десяти строк (можно задать и больше, но они не будут восприняты строкой статуса) с какой-либо дополнительной информацией, описывающей процесс вывода. Первая строка списка будет выведена при наличии в свойстве statusFormat управляющего символа %0, вторая — при наличии символа %1 и т. д.

При помощи перечисленных свойств вы сможете детально описать процесс вывода отчета. В этом вам помогут методы-обработчики событий компонента TRvSystem.

До начата печати отчета и по его окончании (даже если печать была прервана) соответственно вызывается пара методов-обработчиков:

property OnBeforePrint: TNotifyEvent;

property OnAfterPrint: TNotifyEvent;

В начале печати непосредственно отчета (не заголовка) вызывается метод-обработчик

property OnPrint: TNotifyEvent;

Если вы печатаете одну страницу, будет вызван метод-обработчик

type

TPrintPageEvent = function(Sender: TObject;

var PageNum: Integer): Boolean;

property OnPrintPage: TPrintPageEvent;

Но до начала печати вызывается метод-обработчик

property OnNewPage: TNotifyEvent;

который обозначает генерацию страницы.

При печати колонтитулов в верхней и нижней частях страницы вызываются методы-обработчики

property OnPrintHeader: TNotifyEvent;

property OnPrintFooter: TNotifyEvent;

Разработчик может задать несколько опций для всего компонента TRvSystem, управляя тем самым процессом вывода отчета. Для это используется свойство

type

TSystemOption = (soUseFiler, soWaitForOK, soShowStatus, soAl-
lowPrintFromPreview, soPreviewModal);

TSystemOptions = set of TSystemOption;

property SystemOptions: TSystemOptions;

Элементы типа TSystemOptions обозначают следующее:

- soUseFiler — при установке этой опции в значение True вывод будет направляться в файл, заданный свойством SystemFiler, независимо от других настроек компонента;
- SoWaitForOK — если включить эту опцию, генерация отчета будет задержана до момента, когда пользователь нажмет кнопку ОК в диалоге настройки печати компонента (см. рис. 23.4);
- soshowstatus — эта опция управляет видимостью окна состояния процесса вывода отчета в компоненте;
- soAiiowPrintFromPreview — будучи включенной, эта опция позволяет печатать отчет из окна предварительного просмотра;
- soPreviewModal — при значении True делает окно предварительного просмотра модальным.

Таким образом, в качестве основного средства создания отчетов и их использования в приложениях в состав Delphi 7 включен генератор отчетов Rave Reports 5.0. В его состав входят ядро генератора отчетов, визуальная среда создания отчетов и набор компонентов.

Ядро генератора отчетов обеспечивает предварительный просмотр или печать отчета. Оно включается в исполняемый файл приложения. Поэтому разработчики избавлены от необходимости распространять совместно с приложением какие-либо дополнительные файлы.

Визуальная среда создания отчетов позволяет разрабатывать самые разнообразные отчеты, в том числе использующие наборы данных из источников различных типов.

Набор компонентов предоставляет разработчику инструментарий для управления отчетом в приложении.

**Форма контроля: контрольные вопросы**

1. Генератор отчетов Rave Reports 5.0
2. Компоненты Rave Reports и их назначение
3. Отчет в приложении Delphi
4. Компонент отчета TRvProject
5. Компонент управления отчетом TRvSystem

**Рекомендуемая литература**

**Основная литература**

1. Брукс Ф. П. Как проектируются и создаются программные комплексы. М.: Наука, 2009.

2. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.

3. Липаев В.В. Проектирование программных средств. -М.:Высшая школа,1990.

Фокс Дж. Программное обеспечение и его разработка.-М.: Мир, 1985.

4. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.

**Дополнительная литература**

5. Дарахвелидзе П.Г., Марков Е.П. Программирование в Delphi7. Спб.: БХВ - Санкт-Петербург,2003.-784с.:ил.

6. С.Бобровский. Delphi 7. Учебный курс. Издательство «ПИТЕР»,2005г.

7. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».

# СПИСОК ЛИТЕРАТУРЫ


1. Брукс Ф. П. Как проектируются и создаются программные комплексы. М.: Наука, 2009.
2. Хернандес, Майкл Дж. SQL-запросы для простых смертных. – СПб: Питер, 2006.
3. Липаев В.В. Проектирование программных средств. -М.: Высшая школа,2000.
4. Майерс Г. Искусство тестирования программ. - М.: Финансы и статистика, 2002.
5. Фокс Дж. Программное обеспечение и его разработка. - М.: Мир, 1985.
6. Н. Тюкачев, К. Рыбак Программирование в Delphi для начинающих, Издательство: BHV, 2010 г.
7. Фаронов В.В., Шумаков П.В. Delphi. Руководство разработчика баз данных. – М.: «Нолидж», 2001. 640 с.
8. Федоров А., Елманова Н. Введение в базы данных. Питер.: СПБ, 2000.
9. Дарахвелидзе П.Г., Марков Е.П. Программирование в Delphi7. Спб.: БХВ - Санкт-Петербург,2003.-784с.:ил.
10. Смайлова Ә.М. DELPHI ортасында деректер базасын бағдарламалау. Оқу құралы. - Алматы, 2011.
11. С.Бобровский. Delphi 7. Учебный курс. Издательство «ПИТЕР»,2005г.
12. Дарахвелидзе, Е. Марков, О. Котенок. Программирование в Delphi 7. Современные технологии ADO, CORBA, COM. «Издательство BHV-Санкт-Петербург».
13. Методические указания для выполнения лабораторных работ по дисциплине
14. Конспект лекций по дисциплине
15. **Видео-лекции**, ссылка в интернете:
https://www.youtube.com/watch?v=GrhMv7lqaCU - Видео курс SQL Essential (Базовый). Урок 1 Введение в SQL
**https://www.youtube.com/watch?v=bkCWfj-EoUk** - Как создать базу данных в Delphi - создаем клиентское приложение
https://www.youtube.com/watch?v=EUrLjbCd7hU - Создание базы данных и отображение данных.

**Электронные учебники:**

16. http://life-prog.ru/1_1966_osnovi-SQL-kurs-lektsiy.html - Основы SQL. Курс лекций
17. http://gendocs.ru/v31400/%D0%BB%D0%B5%D0%BA%D1%86%Ddelphi - Разработка баз данных на основе Delphi

**Презентации:**

18. **http://presentaci.ru/prezentacii-po-programmirovaniu/75728-postroenie-bazy-dannyh-delphi.html - Построение базы данных Delphi**
19. **http://ppt4web.ru/informatika/postroenie-bazy-dannykh-delpi.html** - базы данных в Delphi