МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

С.Б. Чернова, В.В. Шаптала

Информационные технологии

Лабораторный практикум

Часть 2

Белгород
2015

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

С.Б. Чернова, В.В. Шаптала

Информационные технологии

Лабораторный практикум

Часть 2

Утверждено ученым советом университета в качестве лабораторного практикума для студентов специальностей 230400 – Информационные системы и технологии, 230700 - Прикладная информатика (в менеджменте)

Белгород
2015

УДК 004 (07)
ББК 32.973.26-018.2
Ч419

Рецензенты:
Д-р физ.-мат. наук, проф. Белгородского института развития образования С.Е. Савотченко
Канд. техн. наук, доц. Белгородского государственного технологического университета им. В.Г. Шухова Е.Н. Коробкова

Чернова С.Б.
Ч419 Информационные технологи: лабораторный практикум: учебное пособие: в 2 ч.
Чернова С.Б., Шаптала В.В. – Белгород: Изд-во БГТУ, 2015.-ч.2. – 78 с.

Лабораторный практикум предназначен для изучения современных технологий разработки приложений, реализующих информационную систему на основе базы данных. Информационная система создается в визуальной среде C++Builder.

Лабораторный практикум по курсу «Информационные технологии» предназначен для студентов специальностей 230400 – Информационные системы и технологии, 230700 - Прикладная информатика (в менеджменте).

Публикуется в авторской редакции.

УДК 004(07)
ББК 32.973.26-018.2

Содержание

Введение

Бурное развитие вычислительной техники, потребность в эффективных средствах разработки программного обеспечения привели к появлению систем программирования, ориентированных на так называемую “быструю разработку”. В основе систем быстрой разработки или RAD-систем (Rapid Application Development – среда быстрой разработки приложений) лежит технология визуального проектирования и событийного программирования, суть которой заключается в том, что среда разработки берет на себя большую часть рутины, оставляя программисту работу по конструированию диалоговых окон и созданию функций обработки событий.

Одной из широко используемых RAD-систем является Borland C++Builder, которая позволяет создавать различные программы: от простейших однооконных приложений до программ управления распределенными базами данных. В качестве языка программирования в среде Borland C++Builder используется C++.

Несмотря на появление новых современных технологий и систем программирования, система C++Builder будет устойчиво занимать свою нишу. Это обусловлено меньшей требовательностью к аппаратным ресурсам при разработке приложений, большей легкостью в освоении и применении средств систем для разработки приложений различной степени сложности. Она позволяет создавать приложения с помощью инструментальных программных средств, визуально подготавливать, а также непосредственно писать SQL-запросы к базам данных. С ее помощью можно создавать приложения для работы с локальными и удаленными базами данных.

Лабораторная работа №1

Использование источника данных InterBase в приложениях

После появления первых по-настоящему мощных СУБД (Access, Fox Pro) на непродолжительное время появилась иллюзия, что СУБД способны решать задачи не только по управлению данными, но и любые прикладные задачи. Быстро выяснилось, что языкам программирования, встроенным в СУБД не удастся стать столь же эффективными при решении вычислительных задач, как и универсальные языки программирования (C++ и т.д.). В настоящее время СУБД, а точнее их ядра используются точно по назначению: являются мостом между приложениями (написанными на универсальных языках программирования) и хранимыми данными. Графические оболочки современных СУБД облегчают создание схемы базы данных и решают некоторые другие вспомогательные задачи.

Современные среды разработки, такие как CBuilder, поставляются с большим набором драйверов, содержащих исполняющие ядра СУБД тех или иных форматов. При создании информационной системы эти ядра включаются в исполняемый файл приложения и, таким образом, сами СУБД не требуются.

Современные технологии разработки приложений, реализующих информационную систему на основе базы данных, сводят к минимуму необходимость написания программного кода. В визуальной среде CBuilder можно создать достаточно сложную информационную систему, не написав ни строчки программного кода.

Разработка приложений, использующих источники данных InterBase(Firebird), основана на применении компонентов вкладки InterBase (рис. 1.1.):

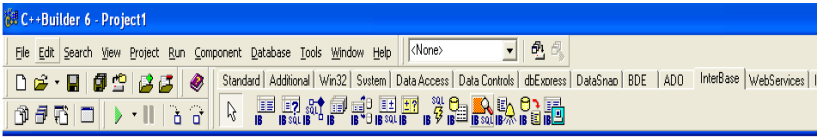


Рис. 1.1. Компоненты доступа к данным по технологии InterBase

При работе с локальными данными используются следующие компоненты:

1. Вкладки InterBase:

IBTable – предназначен для доступа к данным только одной таблицы;
IBQuery - предназначен для выполнения любых запросов к базе данных, как на выборку данных, так и их изменение.
IBTransaction – предназначен для управления транзакцией.
IBDataBase – предназначен для установления соединения с базой данных.

2. Вкладки DataAccess:

DataSource – предназначен для передачи данных от компонентов источников данных (IBTable или IBQuery) к многочисленным компонентам визуализации (графического представления) данных.

3. Вкладки Data Controls:

DBGrid – предназначен для указания сетки или таблицы.

В приложении, использующем базу данных всегда необходимо реализовать следующую информационную цепочку:

БД] IBTable или IBQuery] DataSource] Компоненты визуализации

Рассмотрим процесс построения информационной цепочки:

1. Запустим среду разработки CBuilder 6.0.
2. Изменим заголовок формы со стандартного вида на «InterBase - форма».
3. На форму поместим компоненты IBTable, IBDataBase, IBTransaction, расположенные на вкладке InterBase , а также компонент DataSource .
4. Для просмотра таблицы помещаем на форму визуальный компонент DBGrid вкладки DataControls .
5. С помощью компонента IBDataBase подключим базу, указав в свойстве путь в поле DataBaseName, в свойстве DefaultTransaction устанавливаем IBTransaction1, в свойстве Name устанавливаем IBDataBase1 (рис.1.2).

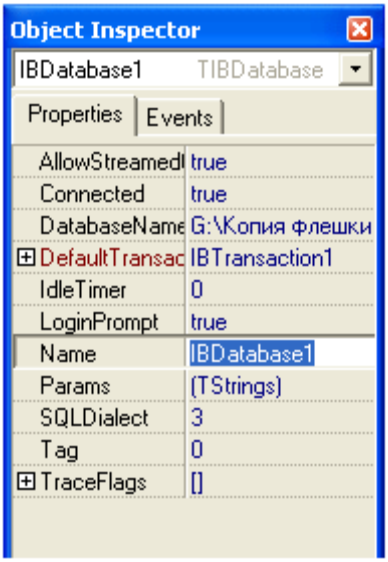


Рис. 1.2. Заполнение свойства DatabaseName, соединяющего с базой данных

6. Для компонента IBTable в свойстве DataBase устанавливаем IBDataBase1, в поле TableName выбираем имя таблицы (рис.1.3).

7. Для компонента IBTransation в свойстве DefaultDatabase устанавливаем IBDatabase1 (рис. 1.4).

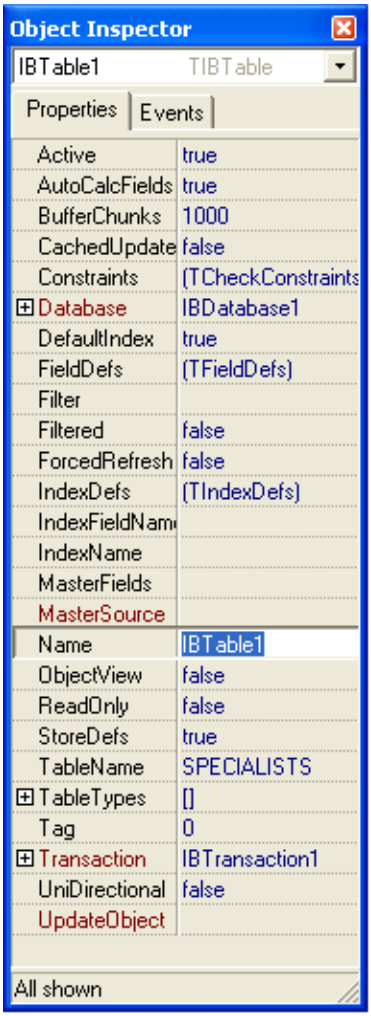


Рис. 1.3. Заполнение свойства IBTableName, соединяющего компонент с таблицей базы данных

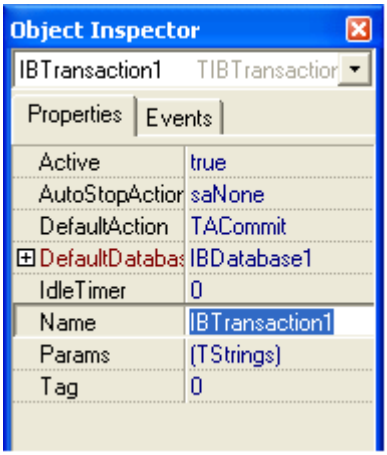


Рис. 1.4. Заполнение свойства DefaultDatabase

8. В свойстве DataSet компонента DataSource устанавливаем IBTable1 (рис.1.5).

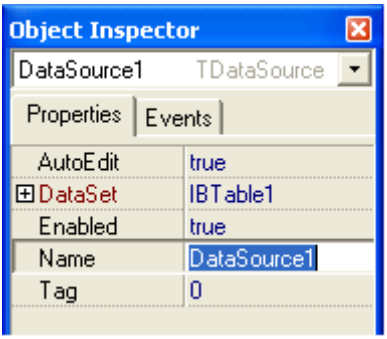


Рис. 1.5. Заполнение свойства DataSet, предназначенного для получения и редактирования данных

9. Визуальное представление набора данных. Подсоединение компонента визуализации DBGrid к источнику данных локальной таблицы.

Визуальное представление данных может осуществляться в различных формах. В одних случаях необходимо целостное, интегрированное представление целого набора данных (множества строк). В других – более подходит детальное представление небольшого фрагмента данных. А часто необходимо сочетать широту охвата с детальным просмотром какой-то определенной части информации. Среда программирования CBuilder предоставляет большой набор компонентов для визуализации данных, которые способны дать нужное разработчику, а что еще более важно, и пользователю представление данных.

Поскольку в среде программирования CBuilder компоненты визуализации данных полностью отделены от компонентов – наборов данных, связанных с источником хранимых данных, то для визуализации данных конкретный тип источника данных (BDE, InterBase или ADO) не имеет большого значения. Каким бы не был набор данных, он всегда соединяется с компонентами визуализации через компонент-шлюз: DataSource.

Все компоненты визуализации расположены на вкладке Data Controls (рис. 1.6):

Рис. 1.6. Компоненты визуализации данных

Компонент DBGrid предназначен для представления большого количества строк из набора данных, полученных с помощью одного из компонентов – наборов (Table, Query или какого-то еще). Другие компоненты визуализации расположенные на вкладке Data Controls могут дать представление только для отдельного поля строки реляционной таблицы.

Чтобы осуществить визуализацию данных отдельной реляционной таблицы, нужно создать следующую цепочку компонентов:

Table/DataSource/DBGrid

и соединить их в единое целое, определив требуемым образом их свойства. Выполним следующую последовательность действий:

Для компонента DBGrid в свойстве DataSource устанавливаем DataSource1 (рис. 1.7).

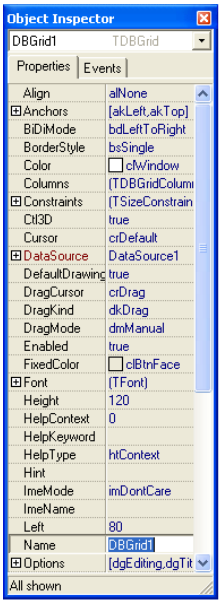


Рис. 1.7. Заполнение свойства DataSource

10. На форму помещаем компонент DBNavigator, расположенный на вкладке DataControls, в свойстве DataSource устанавливаем DataSouce1 (рис.1.8, 1.9).

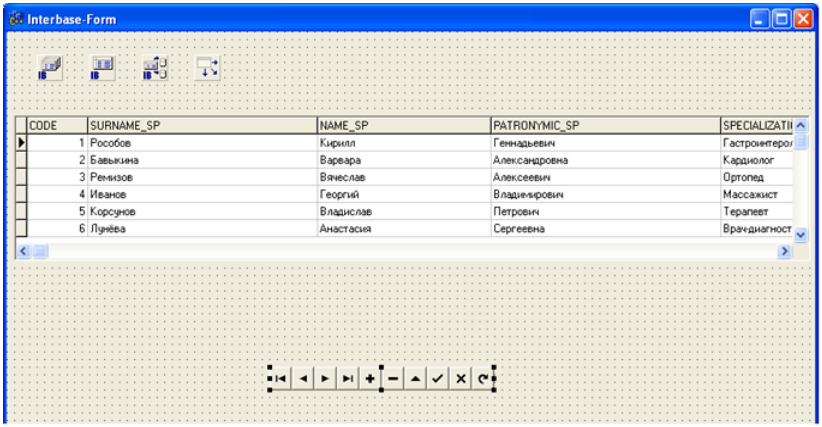


Рис. 1.8. Форма с компонентом DBNavigator

11. Для компонента IBTable свойству Active задаем значение True.

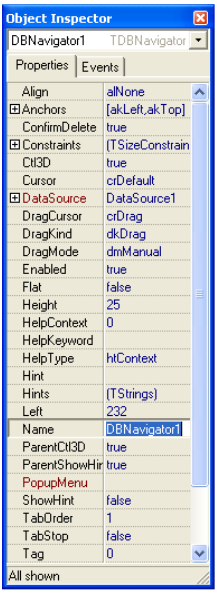


Рис. 1.9. Заполнение свойства DataSource

Если теперь откомпилировать приложение и запустить на выполнение, то оно, будет вполне работоспособным, позволяя перемещаться по строкам таблицы SPECIALISTS.

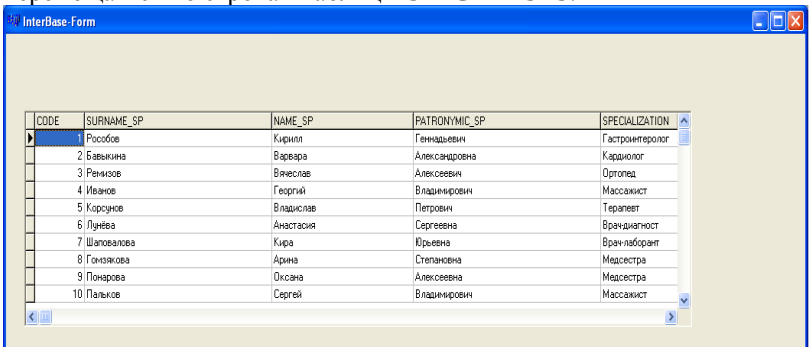


Рис. 1.10. Форма InterBase-Form на этапе выполнения

Лабораторная работа №2

Визуальное представление набора данных

Визуальное представление данных может осуществляться в различных формах. В одних случаях необходимо целостное, интегрированное представление целого набора данных (множества строк). В других – более подходит детальное представление небольшого фрагмента данных. А часто необходимо сочетать широту охвата с детальным просмотром какой-то определенной части информации. Среда программирования CBuilder предоставляет большой набор компонентов для визуализации данных, которые способны дать нужное разработчику, а что еще более важно, и пользователю представление данных.

Поскольку в среде программирования CBuilder компоненты визуализации данных полностью отделены от компонентов – наборов данных, связанных с источником хранимых данных, то для визуализации данных конкретный тип источника данных (BDE, ODBC, ADO и InterBase) не имеет большого значения. Каким бы не был набор данных, он всегда соединяется с компонентами визуализации через компонент-шлюз: DataSource.

1. Подсоединение компонента визуализации DBGrid к источнику данных локальной таблицы. Все компоненты визуализации расположены на вкладке Data Controls (рис. 2.1.):

Рис. 2.1. Компоненты визуализации данных

Компонент DBGrid предназначен для представления большого количества строк из набора данных, полученных с помощью одного из компонентов – наборов (IBTable, IBQuery или какого-то еще). Другие компоненты визуализации расположенные на вкладке Data Controls

могут дать представление только для отдельного поля строки реляционной таблицы.

Чтобы осуществить визуализацию данных отдельной реляционной таблицы, нужно создать следующую цепочку компонентов:

IBTable/DataSource/DBGrid и соединить их в единое целое, определив требуемым образом их свойства. Выполним следующую последовательность действий.

1. Поместим на форму компонент – набор данных IBTable, по умолчанию ему будет присвоено имя IBTable1; сейчас нет большого повода это имя менять, в больших приложениях имя должно быть более осмысленным. Для свойства DatabaseName укажем путь к базе данных. Свойству IBTableName выберем значение SPECIALISTS. Сделав двойной щелчок на свойстве active, установим его в значение true.
2. Поместим на форму компонент DataSource (находится на вкладке Data Access, его имя по умолчанию будет DataSource1) и настроим его свойство DataSet, выбрав для него единственное возможное значение (поскольку других компонентов – наборов данных нет) IBTable1.
3. Наконец, добавим на форму компонент DBGrid (ситема даст ему имя DBGrid1, и его тоже можно не менять) и сначала установим его свойство DataSource (оно будет DataSource1, поскольку других компонентов – источников данных для визуальных компонентов нет), а затем свойству align выберем значение alClient, чтобы компонент DBGrid занял все пространство формы.

Результат представлен на рис. 2.2, на котором показана форма с четырьмя компонентами Table1, DataSource1, Transaction и DBGrid1 на этапе разработки.

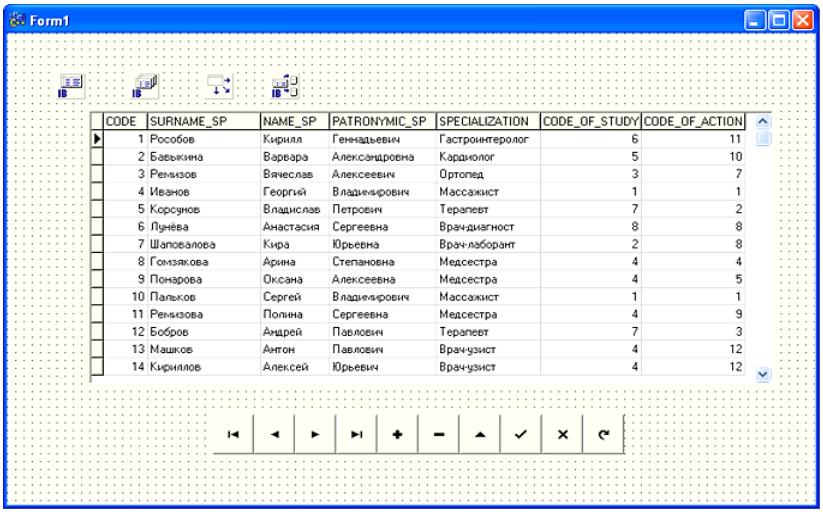


Рис. 2.2. Форма на этапе разработки

Рис. 2.2. показывает, что форма не вполне хорошо выглядит:

не все столбцы содержат полезную информацию, и их не следует выводить;

названия столбцов следовало бы написать русскими буквами.

Рис. 2.3. Пункты контекстного меню компонента DBGrid (вызывается правой кнопкой мыши)

Компонент DBGrid следует отредактировать. Для этого нужно с помощью контекстного меню (рис. 2.3) вызвать редактор столбцов.

Выбрав в контекстном меню компонента DBGrid пункт «Columns Editor…» откроем (вначале пустое) окно редактора (рисунок 2.4).

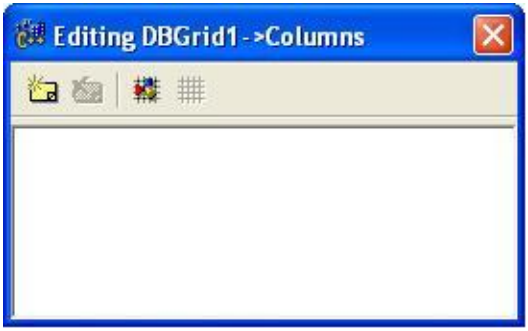


Рис. 2.4. Окно редактора столбцов, непосредственно после вызова из контекстного меню

Первые три слева кнопки на панели инструментов этого окна позволяют: добавить столбец, удалить существующий столбец и добавить столбцы для каждого поля набора данных (рис. 2.5).

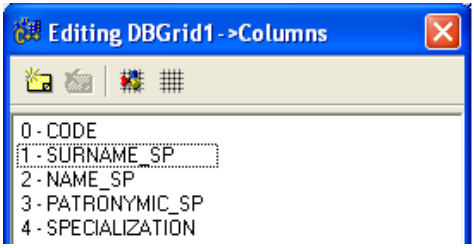


Рис. 2.5. Окно редактора столбцов поле щелчка на третьей кнопке

Теперь осталось удалить ненужный столбец 0. Для этого строка с именем столбца выделяется, и нажимается вторая кнопка. Удаление столбца сразу же изменяет вид компонента DBGrid. Дальнейшее редактирование столбцов производится в инспекторе объектов (рис. 2.6):

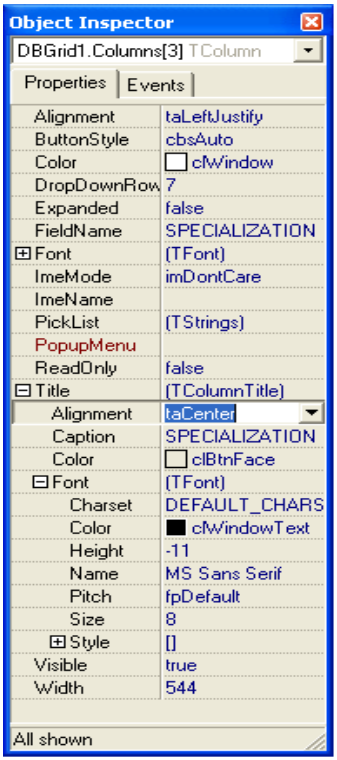


Рис. 2.6. Настройка свойств столбца в инспекторе объектов

Свойство Alignment управляет выравниванием данных столбца (то же свойство внутри комбинированного свойства Title управляет выравниванием в строке заголовка); свойство Caption задает текст заголовка. На рис. 2.7 показана форма до внесения изменений, а на рис. 2.8. после внесения всех изменений: с формы, представленной на рис. 2.7 удалены столбцы, описывающие внутреннюю структуру базы и не имеющие значения для пользователей. Дизайн формы явно выиграл от внесенных изменений. С помощью инспектора объектов можно изменять и другие параметры: шрифт, цвет букв и фона, но

этим не следует увлекаться, поскольку значения по умолчанию всех параметров хорошо подобраны.

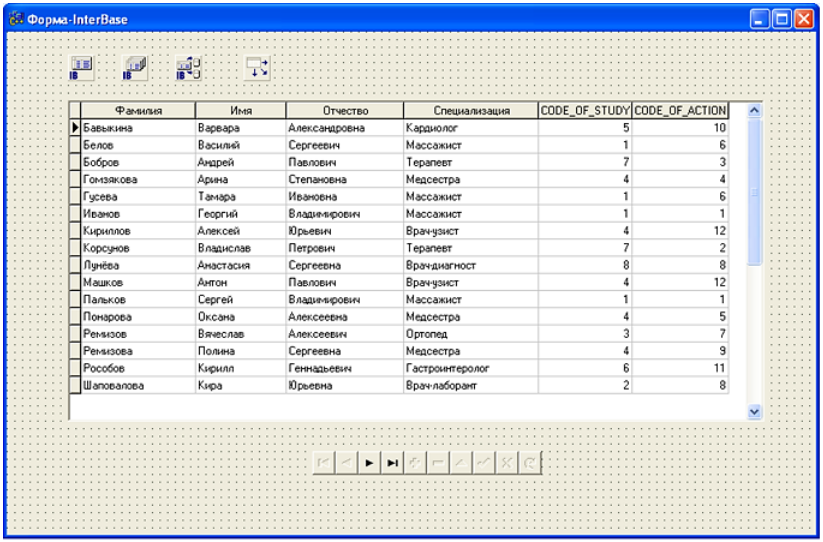


Рис. 2.7. Форма до редактирования компонента DBGrid

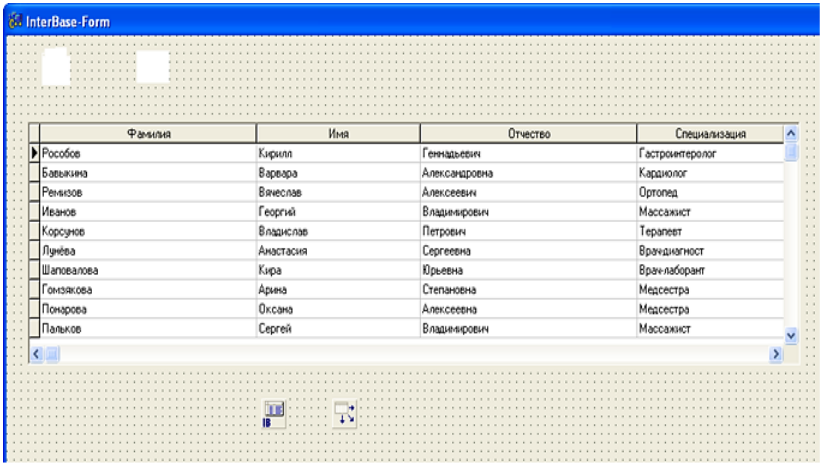


Рис. 2.8. Форма после редактирования компонента DBGrid

2. Сортировка данных. Физически записи в таблицах базы данных расположены в порядке, определяемым примарным ключом таблицы. Напомним, что примарный ключ – это всего лишь поле, или несколько полей, однозначно идентифицирующих запись. Согласно требованиям реляционной модели данных каждая реляционная таблица обязана иметь примарный ключ. В реальных СУБД эти требования несколько ослаблены, и таблица может не иметь примарного ключа. В этом случае, при удалении и добавлении данных, в ней могут появиться пустоты: места в памяти не занятые данными. Чтобы не допустить перерасхода памяти, и нужны примарные ключи. Главные требования к примарным ключам:

минимальная длина;
добавление новых записей в конец таблицы.

Из этого вытекает, что наилучшими кандидатами на роль примарных ключей являются счетчики, последовательно наращивающие свое значение. Однако, такое лишенное всякого смысла физическое упорядочение данных (благоприятное для добавления данных), мало подходит при визуализации данных и их анализе. В этих случаях удобнее для упорядочения данных использовать не примарный ключ (единственный в каждой таблице), а так называемые индексы (или вторичные ключи). При этом достигается множественность (индексов в таблице может быть несколько) способов сортировки записей, с возможностью выбора в конкретной ситуации наиболее удобной.

Сортировка источника данных InterBase. Чаще всего необходимые индексы создаются, при разработке схемы базы данных. В принципе, индексы могут быть добавлены в любое время, но лучше заранее все планировать.

После того, как индекс создан, он может использоваться для сортировки данных.

Откроем новый проект и поместим на форму:

компонент IBTable (имя по умолчанию IBTable1 не будем менять);

компонент DataSource (имя по умолчанию DataSource1 также не будем менять);

компонент DBGrid (имя по умолчанию DBGrid1 также не будем менять).

Установив свойства этих компонентов обычным образом:

для свойства DatabaseName компонента IBTable1 установим путь к базе данных.

свойство TableName компонента IBTable1 установим в значение SPECIALISTS;

свойство DataSet компонента DataSource1 установим в значение IBTable1;

свойство DataSource компонента DBGrid установим в значение DataSource1;

свойство Active компонента IBTable1 нужно сделать равным «true», в результате получим форму, упорядоченную по примарному ключу.

Чтобы записи были упорядочены в алфавитном порядке следует воспользоваться такими способами:

с помощью существующих индексов, для чего нужно задать свойство IndexName (рис. 2.9);

с помощью свойства IndexFieldNames, если индекса нет (рис. 2.10).

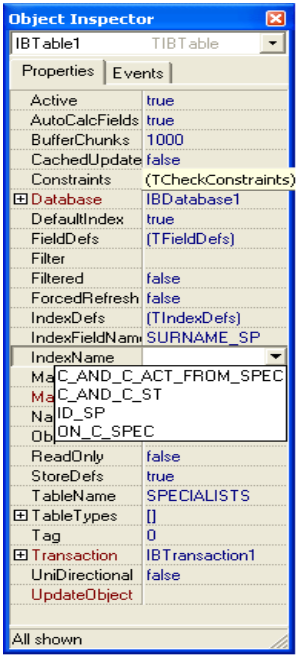


Рис. 2.9. Существующие в таблице индексы можно выбрать в в свойстве IndexName для упорядочения записей

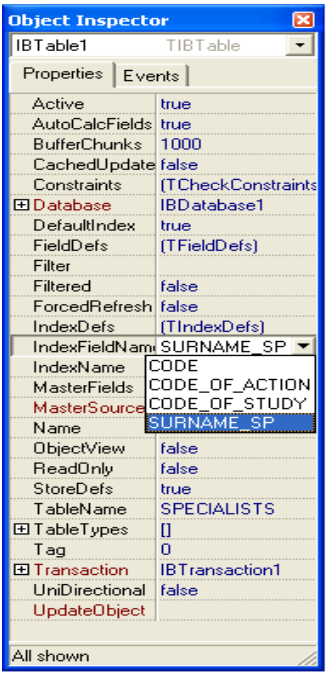


Рис. 2.10. Если необходимого индекса нет, то можно выбрать поля и создать индекс в оперативной памяти

Для сортировки записей таблицы SPECIALISTS по алфавиту выберем, как показано на рис. 2.11, индекс ID_SP – получим требуемый результат: записи в поле Фамилия будут упорядочены в алфавитном порядке.

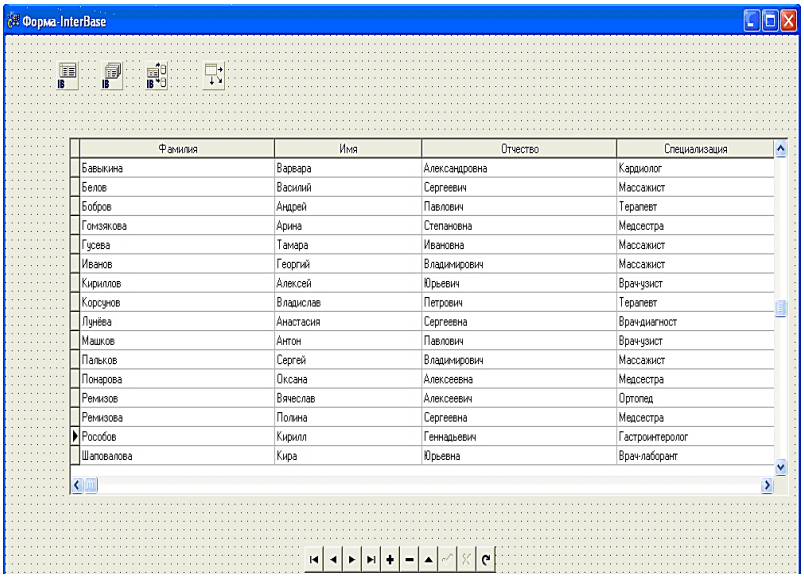


Рис.2.11. После выбора индекса ID_SP записи таблицы SPECIALISTS упорядочены по алфавиту.

Эквивалентным образом можно было бы использовать свойство IndexFieldNames с достижением того же результата, выбрав SURNAME_SP (уже не индекс, а поле), как показано на рисунке 2.10. Стоит заметить, что в списке свойства IndexFieldNames показано поле примарного ключа - n, а списке свойства IndexName присутствует запись PrimaryKey (есть и другие отличия). Применим свойство IndexFieldNames для задания сортировки по двум полям: SURNAME_SP и CODE_OF_STUDY. Тогда записи будут упорядочены по коду кабинета, а взависимости от кода – по алфавиту. Для этого в свойстве IndexFieldNames установим значение «SURNAME_SP; CODE_OF_STUDY» (при задании свойства кавычки ставить не надо). На рис. 2.12 показан результат

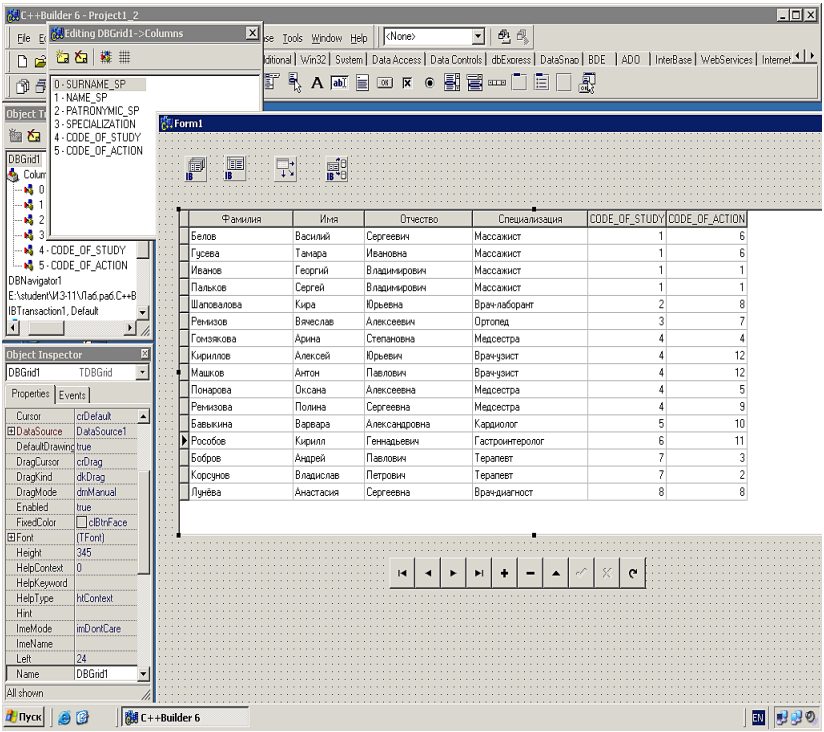


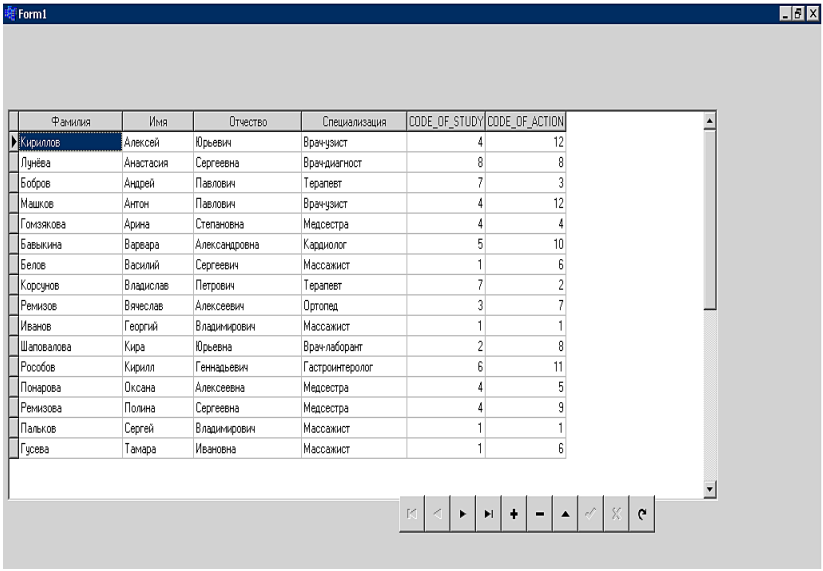
Рис 2.12. Записи упорядочены по коду кабинета, а в зависимости от кода кабинета по фамилиям в алфавитном порядке

Таким образом, свойства IndexName и IndexFieldNames одинаково хорошо решают задачи сортировки записей источников данных InterBase. В рассмотренных примерах порядок сортировки записей устанавливался на этапе разработки, но ничто не мешает делать это на этапе выполнения. Пусть требуется, чтобы при щелчке на заголовке столбца записи упорядочивались по значениям поля, отображаемого в столбце. Для этого достаточно написать обработчик щелчка на заголовке:

```
void_fastcall TForm1::DBGrid1TitleClick(TColumn *Column)
{
  Form1->IBTable1->Active=false;
```

```
  Form1->IBTable1->IndexFieldNames=Column->FieldName;
  Form1->IBTable1->Active=true;
}
```

Обработчик имеет название TForm1::DBGrid1TitleClick поскольку щелчок (Click) осуществляется в строке заголовка (Title) компонента DBGrid1, принадлежащего форме класса TForm1. Единственный аргумент Column является указателем на тот столбец, заголовок которого был отмечен щелчком. В первой строке кода обработчика разрывается связь с источником данных. Так нужно делать всегда при изменении любых параметров источника. Затем новое значение свойства IndexFieldNames устанавливается равным имени того поля, которое отображается в столбце. После этого снова устанавливается соединение с источником данных. Результат работы программы показан на рис. 2.13 и 2.14:



Фамилия	Имя	Отчество	Специализация	CODE_OF_STUDY	CODE_OF_ACTION
Кириллов	Алексей	Юрьевич	Врачгузист	4	12
Лунёва	Анастасия	Сергеевна	Врачдиагност	8	8
Бобров	Андрей	Павлович	Терапевт	7	3
Машков	Антон	Павлович	Врачгузист	4	12
Гомзякова	Арина	Степановна	Медсестра	4	4
Бабыкина	Варвара	Александровна	Кардиолог	5	10
Белов	Василий	Сергеевич	Массажист	1	6
Корсунов	Владислав	Петрович	Терапевт	7	2
Ремизов	Вячеслав	Алексеевич	Ортопед	3	7
Иванов	Георгий	Владимирович	Массажист	1	1
Шаповалова	Кира	Юрьевна	Врач-лаборант	2	8
Рособов	Кирилл	Геннадьевич	Гастроинтеролог	6	11
Понарова	Оксана	Алексеевна	Медсестра	4	5
Ремизова	Полина	Сергеевна	Медсестра	4	9
Пальков	Сергей	Владимирович	Массажист	1	1
Гусева	Тамара	Ивановна	Массажист	1	6

Рис. 2.13. Записи, упорядоченные по Имени

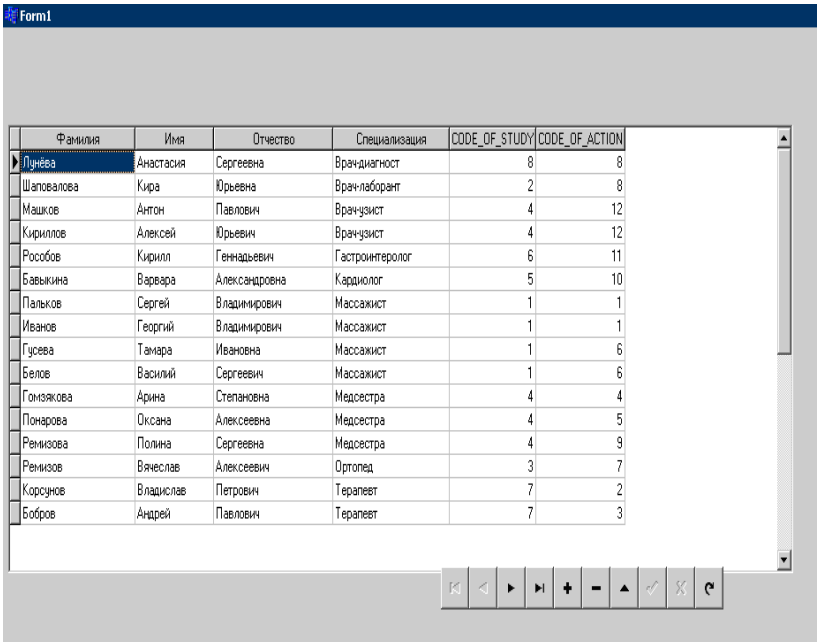


Рис 2.14. Записи, упорядоченные по Специализации

В данном примере проще было использовать свойство IndexFieldNames, но в других случаях может быть удобнее работать непосредственно с имеющимся индексом через свойство IndexName. Если нужного индекса нет, его можно создать в программном коде, а затем использовать. Для этого нужно в коллекцию IndexDefs определений индексов добавить новый индекс с помощью метода AddIndex(…).

В тех случаях, когда записи набора данных BDE сформированы на основе не одной, а нескольких реляционных таблиц, нужно использовать компонент IBQuery. Свойство SQL этого компонента позволяет задать упорядочение данных в конструкции ORDER BY, в которой задаются одно или несколько полей, определяющих порядок следования записей набора данных.

Откроем новый проект, и поместим на форму компоненты:

1. IBQuery (имя по умолчанию IBQuery1), и в его свойстве DatabaseName установим путь к базе данных.

В его свойство SQL поместим следующий SQL – код:

```
Select SURNAME_SP,PATRONYMIC_SP,NAME_SP,TITLE
```

From MEDICAL_ACTIONS,SPECIALISTS
Where CODE_OF_ACTION=MEDICAL_ACTIONS.CODE

Данный SQL – код выводит в наборе данных, связанном с компонентом Query фамилию, имя, отчество специалиста и наименование процедуры из таблицы Title.

2. DataSource (имя по умолчанию DataSource1), в его свойстве DataSet установим значение IBQuery1.

3. DBGrid (имя по умолчанию DBGrid1), в его свойстве DataSource установим значение DataSource1.

Форма примет вид (после обычной настройки свойств столбцов DBGrid1) как на рис. 2.15:

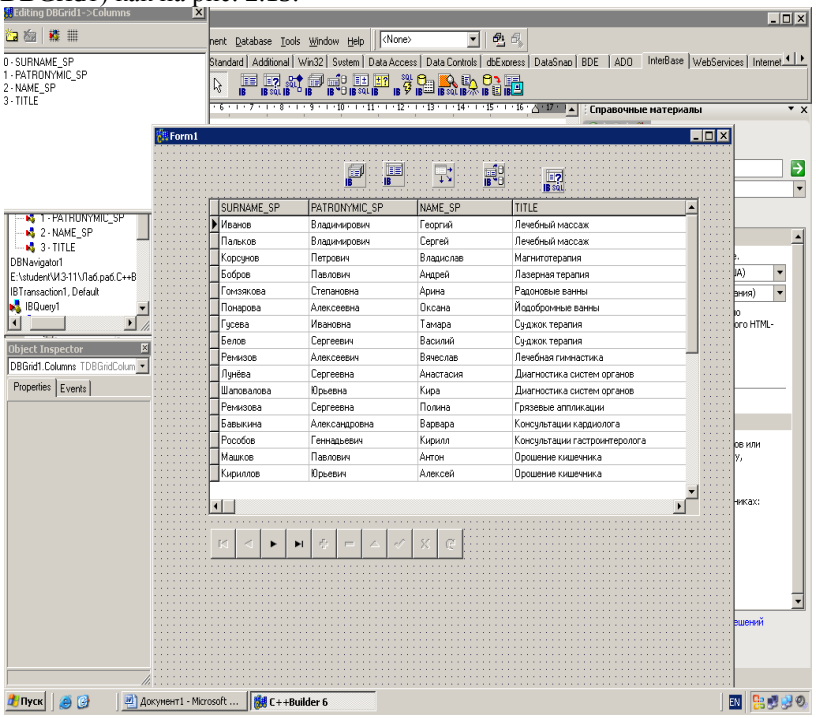


Рис. 2.15. Создание набора данных из двух таблиц MEDICAL_ACTIONS,SPECIALISTS (пока не упорядоченного)

Для набора данных на основе компонента IBQuery нет возможности использовать индексы. У компонента IBQuery, даже, нет

свойств связанных с индексами, как это было у компонента IBTable (свойства IndexDefs, IndexFieldNames, IndexName отсутствуют).

Однако, если в свойство SQL компонента Query добавить строку ORDER BY:

```
Select SURNAME_SP,PATRONYMIC_SP,NAME_SP,TITLE
From MEDICAL_ACTIONS,SPECIALISTS
Where CODE_OF_ACTION=MEDICAL_ACTIONS.CODE
Order By SURNAME_SP
```

где указать поле или список полей, то записи будут уже отсортированы в нужном порядке (рис.2.16.).

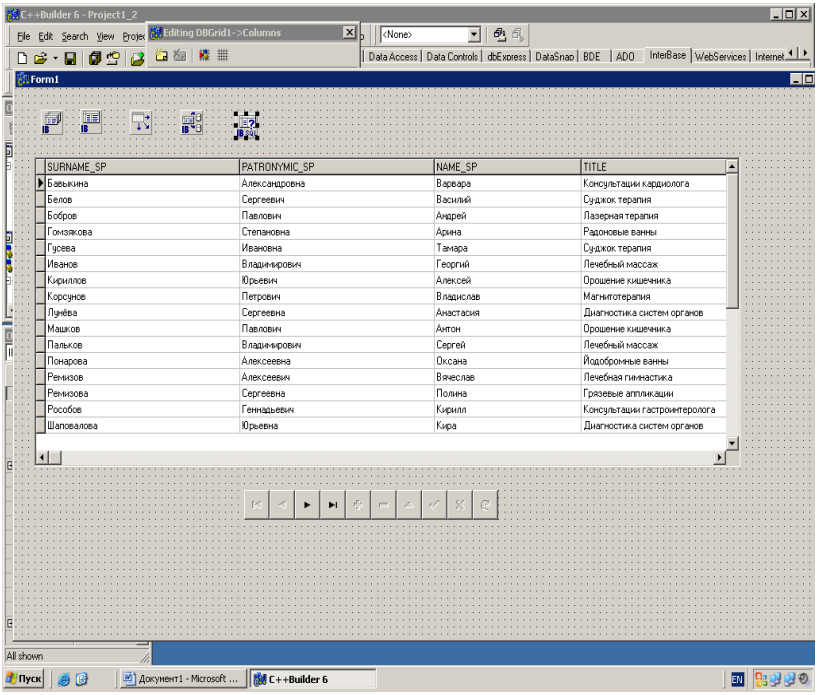


Рис. 2.16. У компонента Query в директиве ORDER BY установлено поле сортировки: SURNAME_SP, и записи располагаются по возрастанию этого поля

Не представляет большого труда менять порядок сортировки на этапе выполнения, если запрограммировать следующий обработчик:

```
void __fastcall TForm1::DBGrid1TitleClick(TColumn *Column)
{
 IBQuery1->Active=false;
 IBQuery1->SQL->Clear();
 IBQuery1->SQL->Add("SELECT
SURNAME_SP,PATRONYMIC_SP,NAME_SP,TITLE");
 IBQuery1->SQL->Add("FROM
MEDICAL_ACTIONS,SPECIALISTS");
 IBQuery1->SQL->Add("WHERE
CODE_OF_ACTION=MEDICAL_ACTIONS.CODE");
 IBQuery1->SQL->Add("ORDER BY " + Column->FieldName);
 IBQuery1->Active=true;
 }
```

Обработчик события onTitleClick уже использовался ранее для компонента IBTable (там он был значительно короче), а в данном случае он выполняет следующие действия:

– компонент IBQuery1 делается неактивным, чтобы было можно изменить его свойство SQL;

– свойство IBQuery1->SQL (типа TStrimgs) очищается перед добавлением новых строк;

– методом Add свойства IBQuery1->SQL добавляется три постоянные строки;

– добавляется Add("ORDER BY " + Column->FieldName) переменная строка, зависящая от того, заголовок какого столбца был отмечен;

– после завершения изменений компонент IBQuery1 делается снова активным.

Данная методика позволяет динамически (во время выполнения программы) менять порядок следования записей набора данных, полученных из нескольких (не обязательно двух) реляционных таблиц с любым набором полей.

3. Фильтрация данных. Фильтрацию данных можно рассматривать в трех аспектах:

– вертикальная фильтрация набора данных, когда в компоненты визуализации передается только часть полей из всех записей набора данных;

– горизонтальная фильтрация набора данных, когда в компоненты визуализации передается только часть записей набора данных со всеми полями;

– комплексная фильтрация набора данных, когда в компоненты визуализации передается часть полей из записей, отобранных по некоторому критерию.

Сам набор записей может формироваться из хранимых данных как одной, так и нескольких таблиц. Вертикальная фильтрация всегда может осуществляться с помощью компонентов визуализации (например, DBGrid) независимо от того, каким компонентом представлен набор данных BDE или InterBase. Поэтому далее основное внимание уделяется горизонтальной (и это специально не оговаривается) фильтрации записей, методика осуществления которой существенно зависит от того, каким является набор данных: получен ли он их одной реляционной таблицы; из двух, связанных по внешнему ключу, таблиц; или по запросу к базе данных. Таким образом, и это важное отличие от вертикальной фильтрации, горизонтальная фильтрация всегда осуществляется в наборах данных, а не в компонентах визуализации (хотя компоненты IBQuery могут осуществлять как горизонтальную, так и вертикальную фильтрацию данных).

Комплексная фильтрация набора данных всегда получается наложением вертикальной фильтрации на горизонтальную фильтрацию.

Фильтрация записей одной таблицы. Задача фильтрации записей одной таблицы возникает естественным образом, когда из большого числа записей, хранящихся в таблице, в компонентах визуализации (чаще всего в компонент DBGrid) нужно показать лишь небольшую их часть. Например, из всего списка студентов желательно показать только записи о студентах одной группы.

Откроем новый проект и поместим на форму компоненты:

1. IBTable (имя по умолчанию Table1), в его свойстве DatabaseName укажем путь к базе данных, подсоединив через свойство TableName к таблице SPECIALISTS, используя источник данных .
2. DataSource (имя по умолчанию DataSource1), его свойству DataSet присвоим значение Table1.
3. DBGrid (имя по умолчанию DBGrid1), его свойству DataSource присвоим значение DataSource1.
4. LabeledEdit (имя по умолчанию LabeledEdit1), его свойству EditLabel|Caption присвоим значение «Введите специализацию».

В компонент IBTable1 добавим все поля, а в компонент DBGrid1 – все столбцы; и произведем настройку необходимых свойств.

Так задавать условия отбора записей можно только для иллюстрации использования свойств Filter и Filtered компонента IBTable, но на практике их нужно задавать и изменять в программном коде.

Можно разными способами менять динамически условия отбора нужных записей. Поскольку в данном примере меняется только специализация, запрограммируем обработчик выхода из компонента LabeledEdit1 после ввода специализации(событие onExit).

```
void __fastcall TForm1::LabeledEdit1Exit(TObject *Sender)
{
  AnsiString str;
  IBTable1->Active=false;
  IBTable1->Filtered=false;
  Str=" SPECIALIZATION=' " + Edit1->Text +  " ' ";
  IBTable1->Filter=str;
  IBTable1->Filtered=true;
  IBTable1->Active=true;
}
```

Код обработчика LabeledEdit1Exit достаточно прост:

– сначала компонент IBTable1 отсоединяется от источника данных – таблицы SPECIALIZATION ;

– свойство Filtered компонента IBTable1 устанавливается в значение false, что означает отсутствие фильтрации;

– в свойство Filter компонента IBTable1 устанавливается нужное значение специальности взятое из компонента LabeledEdit1;

– свойство Filtered компонента IBTable1 устанавливается в значение true, что означает фильтрацию записей (рис. 2.19);

– компонент IBTable1 отсоединяется с источником данных.

Результат работы программы показан на рисунке 2.18.

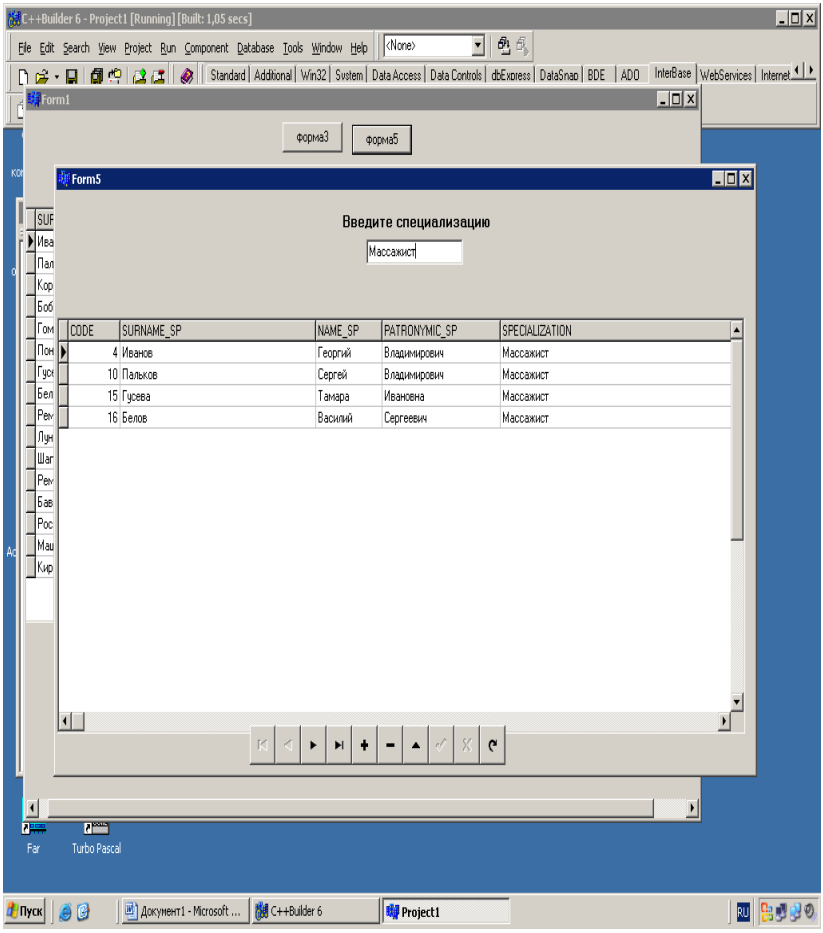


Рис. 2.18. Фильтрация по специализации

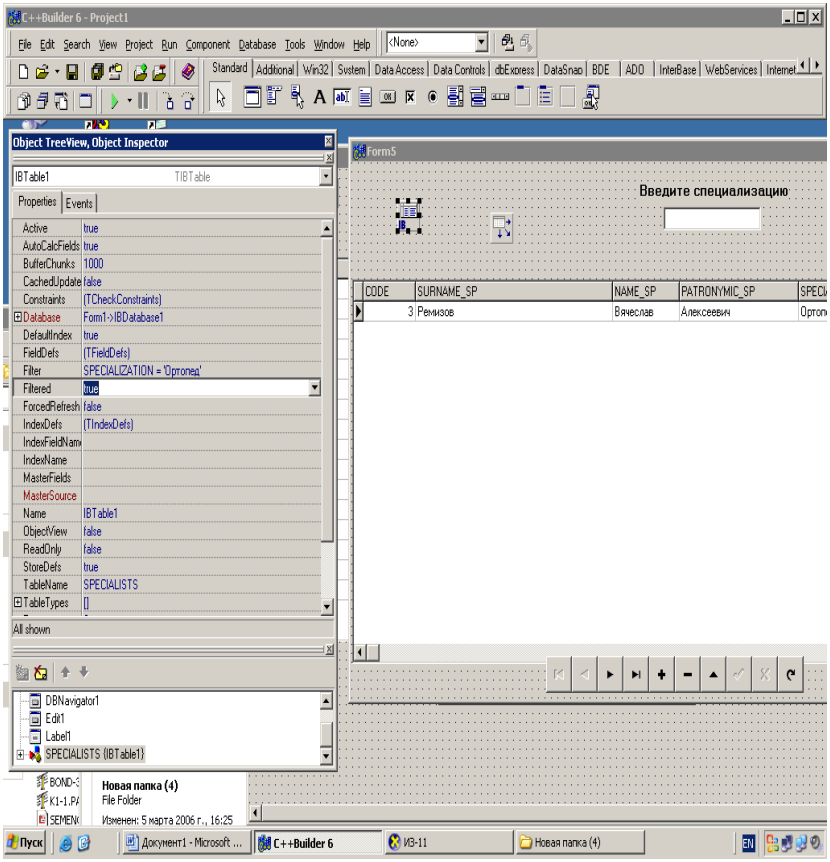


Рис.2.19. Фильтрация с помощью свойства Filtered

Вводя название специальности, каждый раз будем получать записи, только этой специализации, тем самым будет осуществляться горизонтальная фильтрация набора записей реляционной таблицы SPECIALIZATION. Вертикальная фильтрация (например, нужно убрать отображение примарного ключа) достигается средствами самого компонента DBGrid1. Достаточно в редакторе столбцов (вызывается двойным щелчком на компоненте DBGrid1) удалить столбец, в котором отображается примарный ключ (поле n). Этого же результата можно добиться, если:

1. Удалить поле n из набора данных (в редакторе полей компонента IBTable1), но тогда его нельзя будет использовать нигде, что не желательно.

2. У поля n установить в инспекторе объектов свойство visible в значение false, но тогда его нельзя отобразить ни в каких визуальных компонентах, что тоже не желательно.

Таким образом, наилучший вариант вертикальной фильтрации – просто не показывать то или иное поле в визуальном компоненте.

Задание

Создать приложение для своей базы данных, и реализовать сортировку и фильтрацию записей одной таблицы.

Лабораторная работа № 3

Программирование действий с набором записей двух связанных по внешнему ключу таблиц

Подсоединение компонента визуализации DBGrid к данным из двух, связанных по внешнему ключу, таблиц. Информация о специалистах, в том виде как она представлена в таблице SPECIALISTS не является полной: отсутствует точное наименование специализации – вместо нее видны значения примарного ключа из таблицы с наименованиями процедур. Было бы неплохо добавить в эту таблицу, еще один столбец, в котором бы отображалось наименование процедур.

Достигнуть этого с помощью только одного компонента визуализации DBGrid нельзя. Требуется несколько изменить сам компонент набора данных Table1, точнее, к нему нужно добавить новое поле. Откроем новый проект и поместим на форму компоненты DataBase, IBTransaction. С помощью компонента DBGrid создаем два окна просмотра, для каждого окна помещем на форму компоненты IBTable и DataSource (рис.3.1). Соединим указанные компоненты в единую информационную цепочку, как это было сделано ранее:

– у компонента DataBase для свойства DatabaseName указываем путь к базе данных;

– у компонента IBTransaction для свойства DefaultDatabase указываем IBDatabase1;

– у компонента IBTable1 (первое окно) для для свойства Name указываем IBTable1, для свойства Transaction указываем IBTransaction1, а для поля Name – MEDICAL_ACTIONS;

– у компонента DataSource1 свойству DataSet присвоим значение Table1;

– у компонента IBTable1 свойству Active устанавливаем true;

– у компонента IBTable2 (второе окно) для для свойства Name указываем IBTable2, для свойства Transaction указываем IBTransaction2, а для поля Name – SPECIALISTS. Свойству Active устанавливаем true;

– у компонента DataSource2 свойству DataSet присвоим значение Table2;

– у компонента IBTable2 свойству Active устанавливаем true.

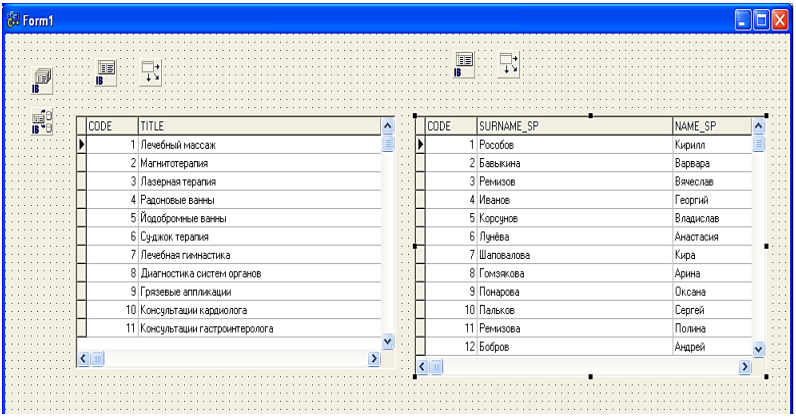


Рис. 3.1. Заготовка формы с двумя таблицами, связанными по внешнему ключу

На следующем этапе будем редактировать компонент набора данных Table1 и Table2 . На рис. 3.2 показано дерево объектов формы в данный момент:

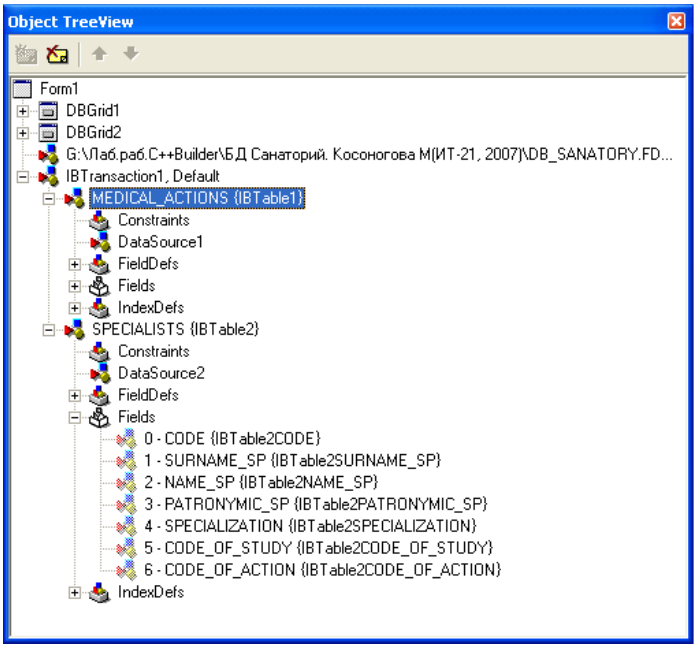


Рис. 3.2 Дерево объектов формы Form1

На рис. 3.2 показаны следующие объекты:

1. База данных DB_SANATORY.
2. Набор данных Table1, содержащий все строки (записи) из таблицы MEDICAL_ACTIONS.
3. Соединенный с набором данных компонент DataSource1.
4. Компонент DBGrid1, содержащий внутри себя объекты столбцов, соответствующих полям реляционной таблицы MEDICAL_ACTIONS.
5. Три коллекции (набора) объектов FieldDefs, Fields и IndexDefs внутри компонента Table1.
6. Набор данных Table2, содержащий все строки (записи) из таблицы SPECIALISTS.
7. Соединенный с набором данных компонент DataSource2.
8. Компонент DBGrid2, содержащий внутри себя объекты столбцов, соответствующих полям реляционной таблицы SPECIALISTS.
9. Три коллекции (набора) объектов FieldDefs, Fields и IndexDefs внутри компонента Table2.

В данный момент наибольший интерес представляет коллекция Fields (рис.3.3), которая содержит поля набора данных. Именно в эту коллекцию будет добавлено поле просмотра (Lookup), в котором будут отображаться значения поля Procedure из таблицы MEDICAL_ACTIONS. Добавление полей осуществляется редактором полей (FildsEditor), вызываемым двойным щелчком мыши на компонентах Table1 и Table2:

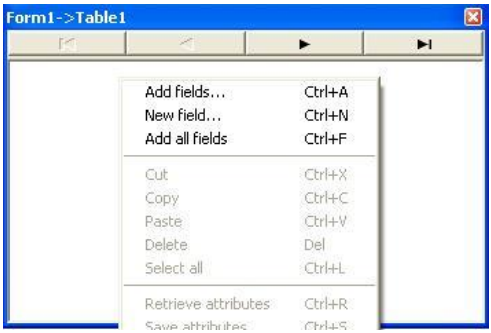


Рис. 3.3 Пустое окно редактора полей набора данных TableN с контекстным меню

Выбрав в контекстном меню пункт Add all fields («Выбрать все поля»), поместим в компонент Table1 все поля реляционной таблицы MEDICAL_ACTIONS , а в компонент Table2 все поля реляционной таблицы SPECIALISTS.

Для всех полей, добавленных в редактор, следует в инспекторе объектов установить некоторые свойства, определяющие некоторые параметры отображения:

– свойство DisplayLabel, определяющее в компоненте DBGrid надпись в заголовке столбца;

– свойство Alignment, определяющее в компоненте DBGrid положение выводимых данных внутри столбца (выравнивание по центру, левому или правому краю).

Добавление поля просмотра. В таблице MEDICAL_ACTIONS есть поле TITLE, а в таблице SPECIALISTS поле CODE_OF_ACTION. В таблицу SPECIALISTS добавим поле просмотра. Для этого вызовем редактор FildsEditor с указанием полей, в контекстном меню выберем пункт «New field…» , после чего откроется окно NewField и произведем заполнение полей, как это показано на рис. 3.4:

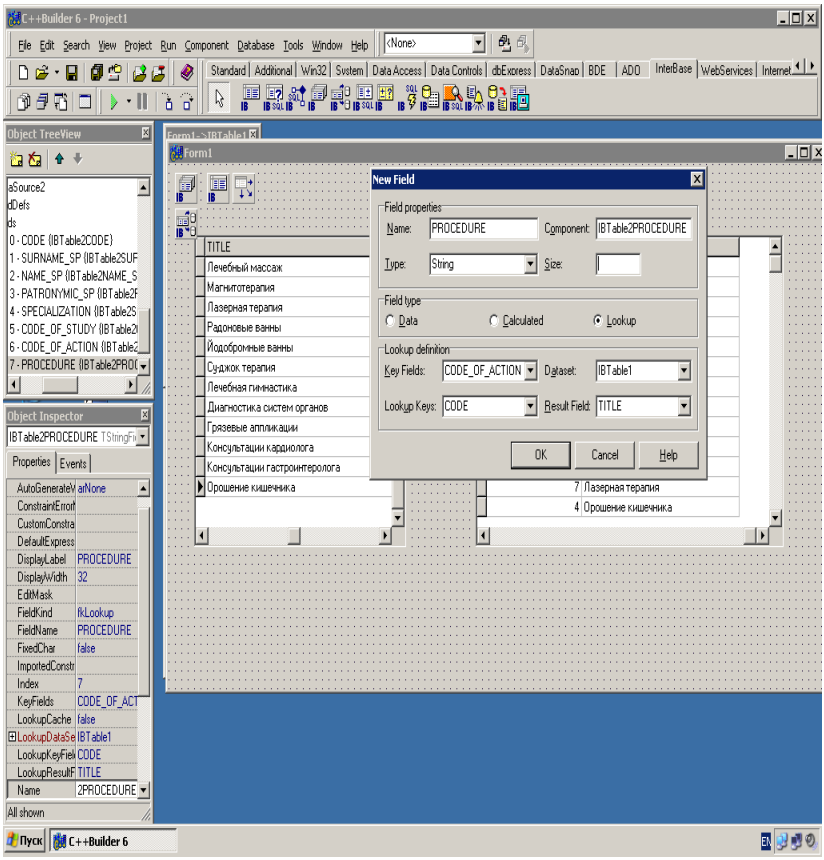


Рис. 3.4 . Окно добавления поля просмотра

В поле Name введено имя добавляемого поля PROCEDURE. В поле Component автоматически будет введено значение Table PROCEDURE 1 . В поле Type нужно выбрать тип поля (в данном случае Sting), и можно указать его размер. В группе переключателей нужно указать какое поле будет добавлено. Значение Data выбирается для реально существующих в таблице полей; значение Calculated – для вычислимых полей (рассчитываются на основе других полей той же таблицы); а значение Lookup – для полей просмотра (которые находятся в связанной по внешнему ключу таблице). После установки переключателя в значение Lookup, появляется возможность выбора в группе Lookup definitions. Здесь в выпадающих списках нужно выбрать:

– поле внешнего ключа Key Fields → CODE_OF_ACTION;

– внешнюю таблицу Dataset (в ней находится поле для просмотра) →IBTable1;

– примарный ключ внешней таблицы Lookup Keys(просматриваемый ключ) → CODE;

– поле для просмотра в DBGrid Result Field (поле, откуда берутся данные в новое поле)→ TITLE;

после подтверждения выбора и закрытия окна будет добавлено новое поле Group, которое можно увидеть в редакторе полей.

Примечание. Свойство Active компонента IBTable2 , в который добавляется новое поле (NewFields) должен иметь значение False (рис.3.5).

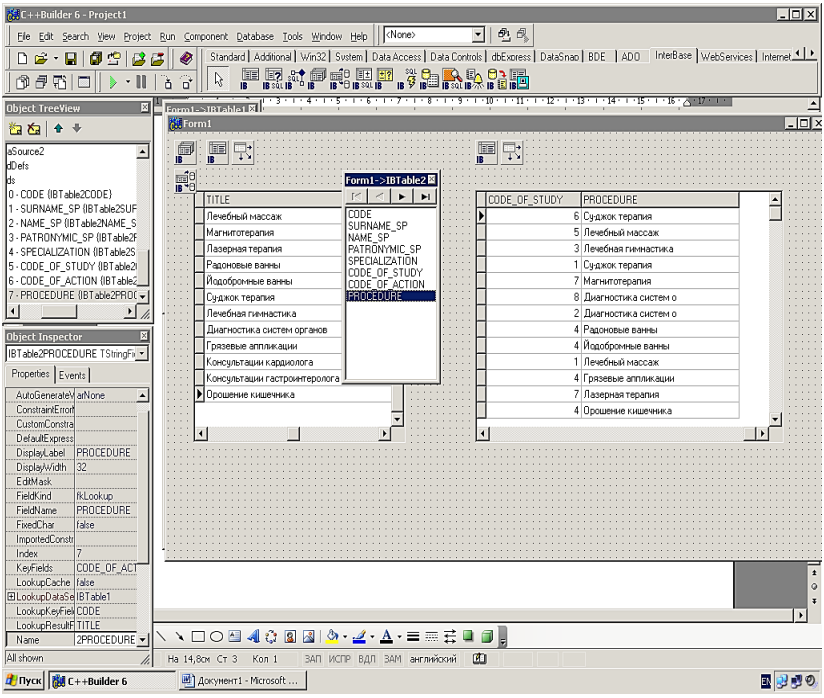


Рис.3.5. Новое поле PROCEDURE добавлено в набор данных IBTable2

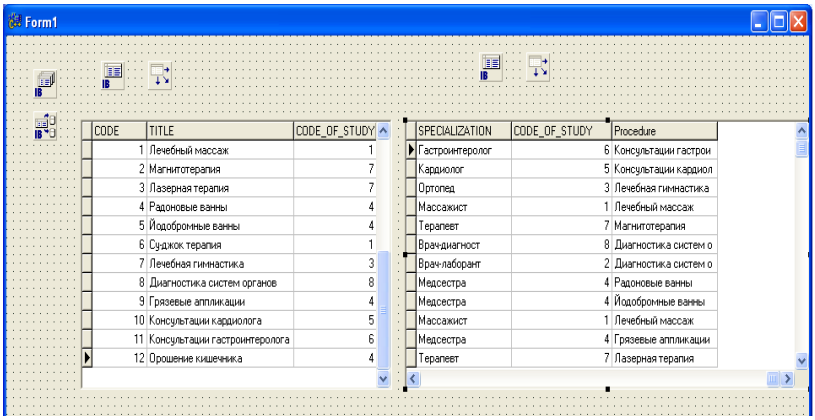


Рис. 3.6. Новое поле PROCEDURE добавлено в компонент DBGrid

Следует отметить, что к набору данных можно добавлять произвольное количество полей просмотра, и не обязательно только из одной внешней таблицы (рис.3.6).

Фильтрация записей связанных по внешнему ключу таблиц. Полезная особенность компонентов Table состоит в том, что в них встроен механизм фильтрации записей по внешнему ключу: в подчиненной таблице можно отфильтровать только те записи, которые отвечают некоторому значению внешнего ключа. При этом значения внешнего ключа (совпадают со значениями примарного ключа главной таблицы) могут отображаться в визуальном компоненте (DBGrid, например) и интерактивно изменяться, меняя и набор отфильтрованных записей.

Откроем новый проект и поместим на форму следующие компоненты:

1. Компонент DataBase, для свойства DatabaseName указываем путь к базе данных.
2. Компонент IBTransaction, для свойства DefaultDatabase указываем IBDatabase1.
3. Table (имя по умолчанию Table1), свойство TableName установим в значение MEDICAL_ACTIONS.
4. DataSource (имя по умолчанию DataSource1), свойство DataSet установим в значение Table1.
5. DBGrid (имя по умолчанию DBGrid1), свойство DataSource установим в значение DataSource1.

6. Table (имя по умолчанию Table2), свойство TableName установим в значение SPECIALISTS.

7. DataSource (имя по умолчанию DataSource2), свойство DataSet установим в значение Table2.

8. DBGrid (имя по умолчанию DBGrid2), свойство DataSource установим в значение DataSource2.

Так настроенные компоненты работают независимо друг от друга, но это не то, что от них требуется. Поэтому у подчиненной таблицы Table2 свойство MasterSource установим в значение DataSource1, установив связь между таблицами. Для того, чтобы установить связь по внешнему ключу, нужно дать правильное значение полю MasterFields (рис. 3.7), которое определяет поля внешнего ключа (их может быть несколько). Данное свойство заполняется не вручную, а в специальном окне (раскрывается кнопкой с многоточием):

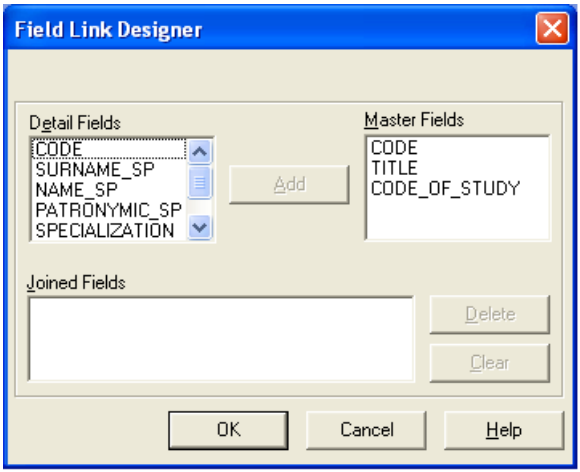


Рис. 3.7.Окно формирования свойства MasterFields

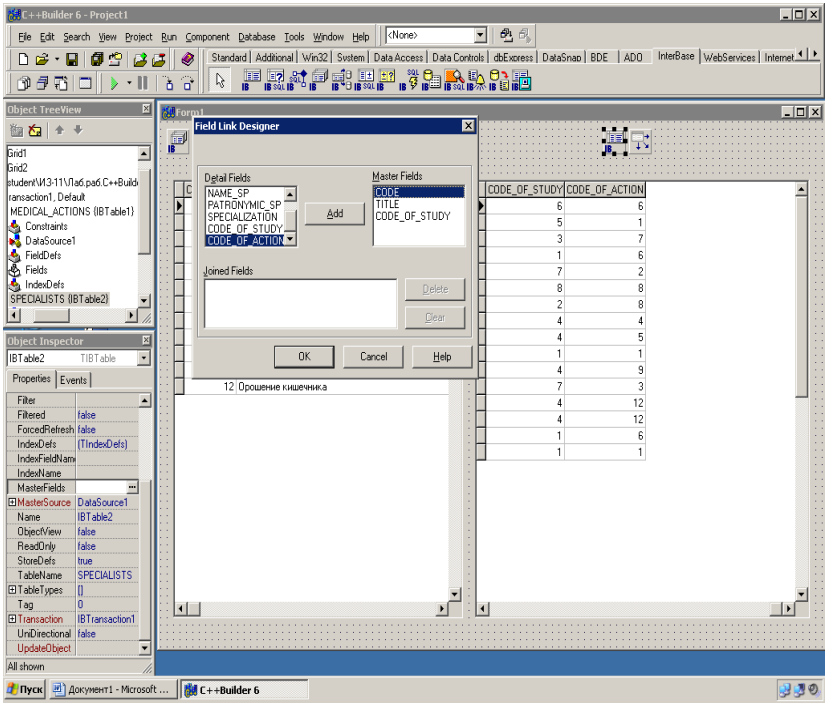


Рис.3.8. Окно формирования свойства MasterFields с выбранными полями

Выполним следующие действия (рис.3.8):

– в окне DefailFields выберем CODE_OF_ACTION;

– в окне MasterFields , выбираем поле, на которое ссылается поле CODE (табл. MEDICAL_ACTIONS);

– кнопкой Add добавляем поля связи;

– запускаем приложение (Run→Run).

Примечание: В данном приложении таблица MEDICAL_ACTIONS является главной, а таблица SPECIALISTS является подчиненной. Передвигаясь по главной форме, в подчиненной получаем отфильтрованные записи.

Новый способ фильтрации обладает несомненными достоинствами:

– на форме нет лишних интерфейсных элементов;

– все возможности выбора фильтра легко просматриваются;

– значения фильтра не нужно вводить, что может привести к ошибкам;
– не понадобилось писать программный код.

Единственный, но существенный недостаток в том, что критерий фильтрации определяется связью таблиц по внешнему ключу; никакой другой критерий фильтрации таким способом реализовать нельзя. Зато каскад, связанных по внешнему ключу таблиц, может быть произвольной длины.

Выбор процедуры из ниспадающего списка. Откроем новый проект и поместим на форму следующие компоненты:

1. Компонент DataBase, для свойства DatabaseName указываем путь к базе данных.
2. Компонент IBTransaction, для свойства DefaultDatabase указываем IBDatabase1.
3. DBGrid (имя по умолчанию DBGrid1).
4. DBLookupComboBox (имя по умолчанию DBLookupComboBox1).

Для компонента DBGrid1 помещаем на форму компоненты IBTable(имя по умолчанию IBTable1) и DataSourse (имя по умолчанию DataSourse1). Для каждого из этих компонент указываем свойства:

IBTable1:
Name → Table1
TableName → SPECIALISTS
Transaction → Transaction1
MasterSourse → DataSourse1
MasterFields → CODE (окно FieldLinkDesigner)
IndexFieldName → CODE_OF_ACTION

DataSourse1:
DataSet → → IBTable1
Name → DataSourse1

DBGrid1:
DataSourse →DataSourse1.

Для компонента DBLookupComboBox1 помещаем на форму компоненты IBTable(имя по умолчанию IBTable2) и DataSourse (имя

по умолчанию DataSourse2). Для каждого из этих компонент указываем свойства:

IBTable2:

Name → Table2

TableName → MEDICAL_ACTIONS

Transaction → IBTransaction1

DataSourse2:

DataSet → IBTable2

Name → DataSourse2

Для компонента DBLookupComboBox1:

ListSourse → DataSourse2

ListField →TITLE

KeyField →CODE

Запускаем приложение на выполнение.

Задание

Создать приложение для своей базы данных, отображающее записи двух связанных по внешнему ключу таблиц и реализовать фильтрацию.

Лабораторная работа №4

Построение запросов

Введение. Borland C++ Builder обладает широкими возможностями доступа к базам данных. Так как базы данных предназначены не только для хранения, но и для выбора и обработки информации, одним из важнейших аспектов их использования является создание запросов к ним. Поэтому мы рассмотрим, как в C++ Builder решаются проблемы построения запросов.

Запрос в C++ Builder - это объект, представляющий собой набор данных. Обычно для создания запроса используется компонент IBQuery - потомок абстрактного класса IBDataSet.

Компонент IBQuery. Компонент IBQuery, как и компонент IBTable, обладает всеми свойствами компонента IBDataSet.

Как и в случае с компонентом IBTable, компонент IBDataSource управляет взаимодействием между компонентами Data Controls и

компонентом IBQuery. Обычно приложение имеет один компонент DataSource для каждого компонента IBQuery.
Наиболее часто используются следующие свойства компонента IBQuery:

Active - указывает, открыт (true) или закрыт (false) данный запрос.

Eof, Bof - эти свойства принимают значение true, когда указатель текущей записи расположен на последней или соответственно первой строке набора данных, являющегося результатом выполнения запроса.

DatabaseName - имя каталога удаленной БД, к которой осуществляется запрос.

DataSource - указывает источник данных для параметризованных запросов (т.е. запросов с параметрами, значение которых заранее неизвестно).

Fields - это свойство доступно только во время выполнения (run-time only) и используется для чтения или модификации поля, определяемого по порядковому номеру.

Params - содержит параметры для параметризованного запроса, как A в следующем примере:

Select * from MEDICAL_ACTION where TITLE=:A

SQL - строковый массив, содержащий текст оператора запроса SQL.

Отметим, что язык запросов SQL (Structured Query Language), традиционно применяемый при работе с серверными СУБД, может быть использован и при работе с таблицами формата dBase и Paradox. Не вдаваясь в подробное описание синтаксиса этого языка (с ним можно познакомиться в других источниках, например, в книге М.Грабера "Введение в SQL"), отметим одну его особенность. SQL - язык непроцедурный. На нем можно написать, что нужно получить в результате запроса, но нельзя написать, как это сделать, то есть нельзя описать саму процедуру выполнения запроса. Дело в том, что реализация выполнения тех или иных операторов SQL серверами баз данных может быть различна, и в большинстве случаев неинтересна клиентскому приложению, создаваемому с помощью C++ Builder. В случае таблиц dBase или Paradox реализацию SQL берет на себя библиотека Borland Database Engine.

Компонент IBQuery позволяет использовать операторы SQL для того, чтобы определять или создавать наборы данных, которые можно отобразить на экране, вставлять, удалять и редактировать строки.

RequestLive - если это свойство имеет значение true и синтаксис запроса таков, что его результат может быть модифицируем, пользователь может редактировать данные с сохранением их в базе

данных. Если RequestLive имеет значение false, результат запроса возвращается в состоянии read-only.

Наиболее часто используются следующие методы компонента IBQuery:

1. ExecSQL - выполняет SQL-запрос, содержащийся в свойстве SQL, если запрос не возвращает данные. Следует употреблять этот метод при вставке, редактировании или удалении данных. При выполнении же оператора SELECT (выбор данных) следует использовать метод Open.

2. Open - открывает компонент IBQuery. Он эквивалентен присвоению свойству Active значения true. Используется, если результатом запроса является набор данных (такие запросы обычно начинаются с оператора SELECT). Пример использования метода Open:

Query1->Open();

3. Close - закрывает компонент IBQuery. Вызов Close эквивалентен присвоению свойству Active значения false. Пример использования метода Close:

Query1->Close();

Компоненты IBQuery обладают большим разнообразием методов, унаследованных от IBDataSet. Наиболее часто используются следующие методы:

1. First, Last, Next, Prior перемещают указатель текущей записи на первую, последнюю, следующую и предыдущую записи соответственно.

2. MoveBy перемещает указатель на определенное количество строк.

3. Insert, Edit, Delete, Append, Post, Cancel - позволяют модифицировать результат запроса. Метод Insert позволяет вносить в результат запроса строки.

Метод Post подтвержает операции Insert, Update или Delete, совершая реальное физическое изменение в базе данных.

Метод Cancel отменяет незавершенные операции Insert, Delete, Edit или Append.

Методы FreeBookmark, GetBookmark, GotoBookmark- - позволяют создавать закладки (маркированные строки) в запросе и затем вернуться к такой строке позже.

Запросы с параметром. Создадим проект, в котором будем выбирать процедуру по ее названию, используя запрос с параметром. Поместим на форму компоненты IBDataBase, IBQuery, TDataSourse и DBGrid (рис.4.1).

Для компонента IBQuery пишем SQL запрос:

Select * from MEDICAL_ACTION where TITLE=:A

В инспекторе объектов в свойстве Params указывается параметр 0-A. Для параметра A в свойстве Value можно задать его значение.

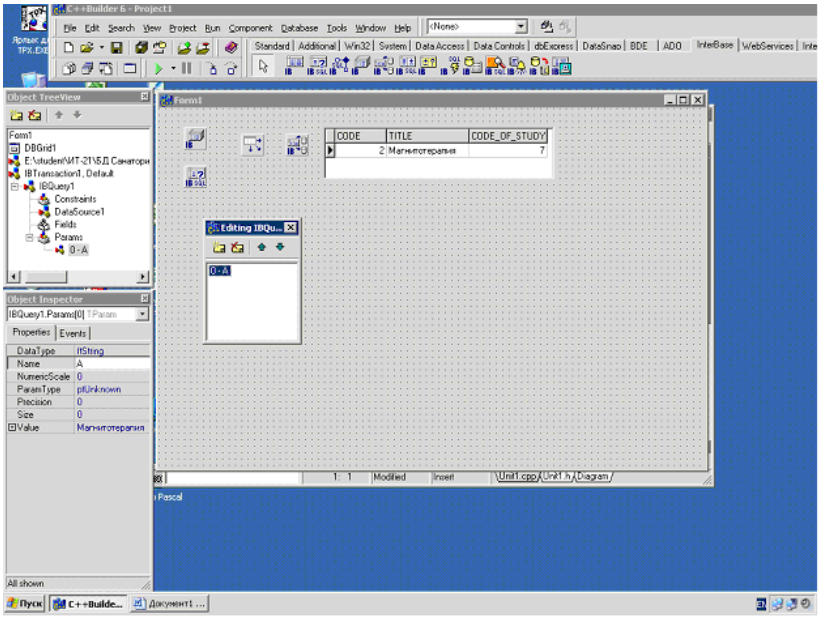


Рис.4.1. Форма с компонентами для выполнения запроса с параметром

Программный доступ к параметрам запроса. Создадим новый проект (рис.4.2). Для выполнения программного доступа к параметрам запроса на форму, кроме указанных выше компонентов, поместим также компоненты Label, Edit и Button (вкладка Standard). Обработчик щелчка кнопки «Выбор» имеет следующий код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 //  IBQuery1->Active = false;
    IBQuery1->Close();
    IBQuery1->Params->ParamByName("A")->AsString= Edit1->Text;
```

```
  //IBQuery1->Active = true;
   IBQuery1->Open();
}
```

В этом коде указаны сразу два варианта программного доступа к параметру запроса (с использованием свойства Active и с использованием методов Open и Close).

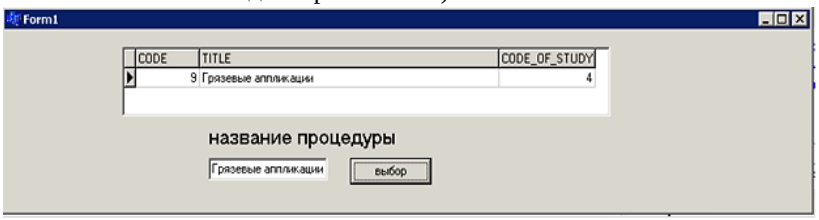


Рис. 4.2. Форма программного доступа к параметру запроса на этапе выполнения

Запросы на изменение.Запрос на удаление. Откроем новый проект и поместим на форму следующие компоненты (рис. 4.3):
В этом проекте выполним запрос на удаление.

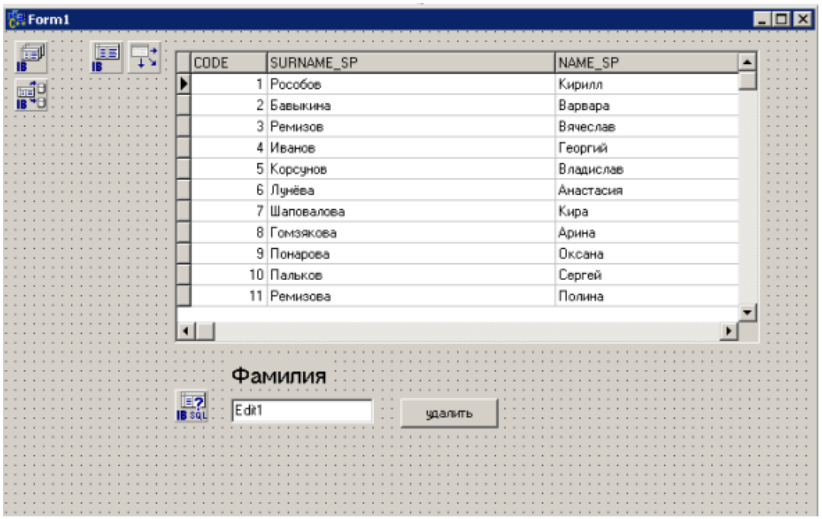


Рис. 4.3. Форма с компонентами, позволяющими выполнить запрос на удаление на этапе разработки

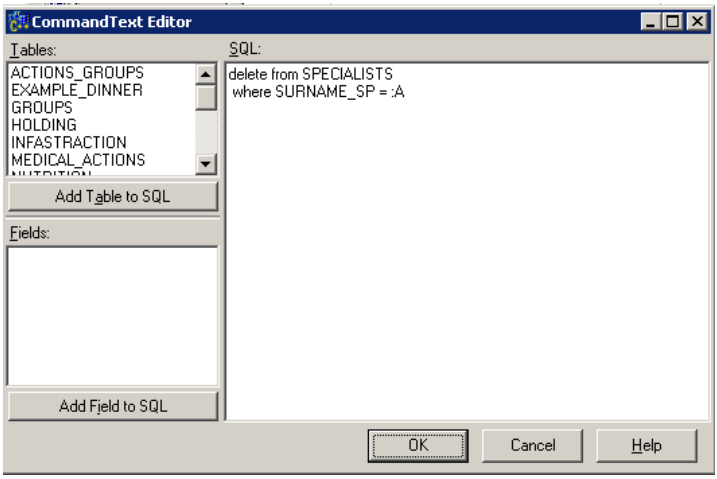


Рис. 4.4. Окно текстового редактора

Обработчик щелчка кнопки «Удалить» имеет следующий код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    IBQuery1->Close();
    IBQuery1->Params->ParamByName("A")->AsString = Edit1->Text;
    IBQuery1->ExecSQL();
    IBTable1->Refresh();    // обновить данные в Table1
 }
```

Результат выполнения запроса (рис.4.5):

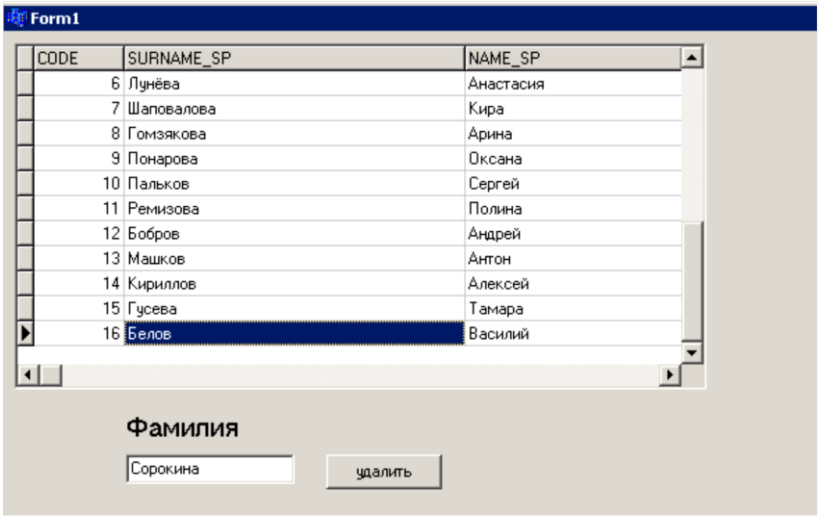


Рис. 4.5. Форма, реализующая запрос на удаление на этапе выполнения

Запрос на добавление. Создадим новый проект. На форму поместим следующие компоненты (рис. 4.6).

В этом проекте реализуем запрос на добавление нового специалиста.

Для компонента IBQuery пишем SQL запрос:

select * from SPECIALISTS

При нажатии кнопки Добавить должен реализовываться следующий обработчик событий:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    IBQuery1->Insert();
    IBQuery1->Fields->FieldByNumber(1)->AsString = Edit1->Text;
    IBQuery1->Fields->FieldByNumber(2)->AsString = Edit2->Text;
    IBQuery1->Fields->FieldByNumber(3)->AsString = Edit3->Text;
    IBQuery1->Fields->FieldByNumber(4)->AsString = Edit4->Text;
    IBQuery1->Fields->FieldByNumber(0)->AsInteger = 100;
    IBQuery1->Fields->FieldByNumber(5)->AsInteger = 4;
    IBQuery1->Fields->FieldByNumber(6)->AsInteger = 4;
    IBQuery1->Post();
     }
```

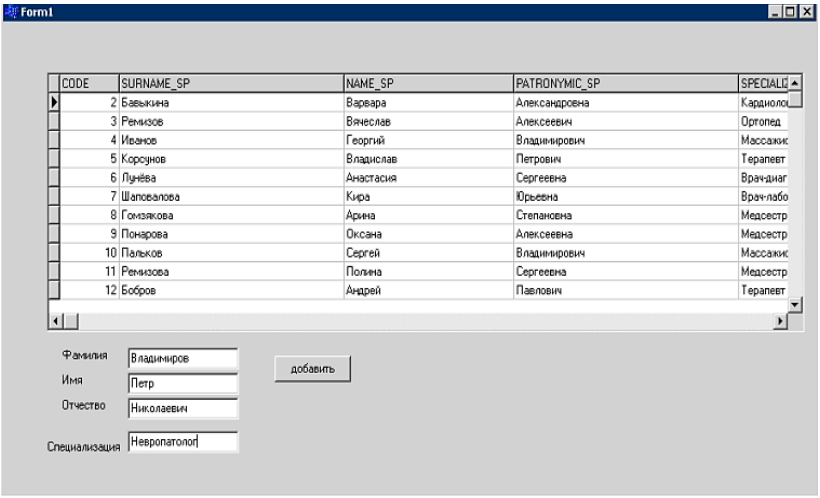


Рис.4.6. Форма, реализующая запрос на добавление на этапе выполнения

Механизм поиска в наборах данных, полученных по запросу. Индексные механизмы поиска не работают в наборах записей, полученных с помощью запроса к нескольким таблицам базы данных. В этом случае для поиска нужной записи используется метод Locate, который не требует в своей работе наличия индекса, что сказывается при поиске в больших массивах данных. Данных метод применим и в случае набора данных на основе компонента Table, но менее эффективен. Метод имеет следующий заголовок:

```
bool __fastcall Locate( const System::AnsiString KeyFields,
const System::Variant &KeyValues, TLocateOptions Options);
```

Следует отметить, что метод Locate может осуществлять поиск в режиме как полного, так и частичного совпадения. Режим определяется третьим параметром, имеющим тип TLocateOptions. Данный тип представляет класс, для которого переопределен оператор «<<», который обычно используется для вывода данных на экран. Если в этот аргумент ввести значение loPartialKey, то поиск будет производиться по частичному совпадению (что используется далее).

Создадим новый проект. В этом проекте реализуем запрос, позволяющий осуществить поиск специалиста по фамилии (рис.4.7). Обработчик щелчка кнопки «Найти» имеет следующий код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TLocateOptions LO;
IBQuery1->Locate("SURNAME_SP",Edit1->Text,
LO<<loPartialKey<<loCaseInsensitive);
    }
```

Подводя итог, можно отметить наличие в среде разработки CBuilder развитых средств сортировки, фильтрации и поиска данных, которые сводят к минимуму создание программного кода. Большая часть свойств компонентов настраивается в инспекторе объектов или в специальных редакторах на этапе разработки.

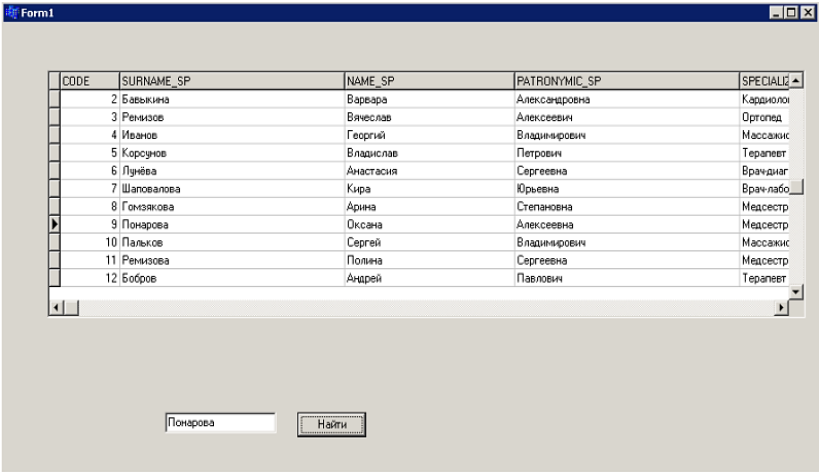


Рис. 4.7. Поиск специалиста по фамилии

Задание

В созданном приложении реализовать необходимые запросы соответствующие предметной области.

Лабораторная работа №5

Хранимые процедуры

Введение. Хранимая процедура представляет собой программу, расположенную на сервере и вызываемую из приложения клиента. Использование этих объектов увеличивает скорость доступа к БД по следующим причинам:

вместо текста запроса, который может быть достаточно длинным, серверу передается по сети относительно короткое обращение к хранимой процедуре;

хранимая процедура, в отличие от запроса, не требует предварительной синтаксической проверки.

Виды хранимых процедур. По числу строк, возвращаемых в качестве результата, можно выделить следующие виды хранимых процедур:

– процедуры выбора, которые возвращают несколько строк – передают набор данных, записями в котором являются строки результатов;

– исполняемые процедуры, которые возвращают одну строку и обеспечивают тем самым возврат значений выходных параметров.

В теле процедуры выбора размещаются совместно используемые операторы выбора нескольких записей и возврата значений – FOR SELECT … DO … SUSPEND, которые и обеспечивают отбор требуемых записей и построчную передачу значений их столбцов в точку вызова.

Вызов хранимой процедуры представляет собой обращение к процедуре с передачей исходных данных для обработки и последующим получением результатов. Исходные данные передаются через входные параметры, а результаты возвращаются через выходные параметры. В зависимости от вида хранимой процедуры различаются способы ее вызова.

Вызов хранимых процедур выбора. Хранимая процедура выбора, как правило, вызывается с помощью оператора выбора SELECT, внутри которого размещается вызов процедуры (рис. 5.1).

Например: создадим приложение для базы данных Санаторий. На форму поместим компоненты IBDataBase, IBTransaction, IBQuery (вкладка InterBase) , компонент DataSource (вкладка DataAccess) и компонент DBGrid (вкладка Data Controls) (рис. 5.2).

Для компонента IBQuery пишем SQL запрос:

SELECT * FROM LIST_OF_NUTR_FROM2(1)

Здесь вызываемая хранимая процедура LIST_OF_NUTR_FROM2 возвращает наименование блюд, указанных в меню на завтрак, обед, полудник и ужин.

Хранимая процедура LIST_OF_NUTR_FROM2 имеет следующий код:

```
BEGIN
NUMBER = 0;
FOR
   select   NUTRITION.FIRST_BREAKFAST, NUTRITION.DINNER,
NUTRITION.LUNCH, NUTRITION.SUPPER
   FROM NUTRITION
   INNER JOIN GROUPS ON
GROUPS.CODE=NUTRITION.CODE_OF_GROUP
   WHERE CODE_OF_GROUP = : CodeGroup
   into :firstBreakfast, :dinner, :lunch, :supper
  DO
   begin
   NUMBER = NUMBER + 1;
   suspend;
   end
END
```

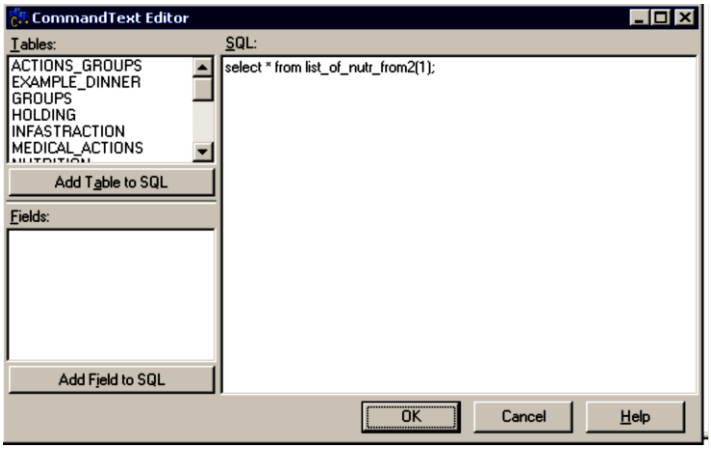


Рис. 5.1. Окно текстового редактора

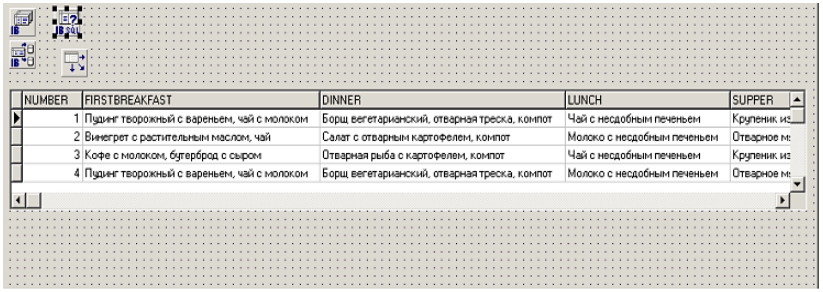


Рис.5.2. Форма, реализующая вызов хранимой процедуры выбора

Вызов исполняемых хранимых процедур. Для вызова исполняемой процедуры из приложения предназначен компонент IBStoredProc. Свойство StoredProcName типа AnsiString определяет вызываемую хранимую процедуру. Имя процедуры выбирается через список Инспектора объектов. После задания имени хранимой процедуры становятся доступными ее входные и выходные параметры, определяемые значением свойства Params типа TParams. Это свойство определяет коллекцию (массив) параметров компонента IBStoredProc, работа с которыми аналогична работе с параметрами SQL-запроса компонента IBQuery. Выполнение выбранной хранимой процедуры осуществляется последовательным вызовом методов Prepare и ExecProc. Метод Prepare подготавливает хранимую процедуру к

выполнению путем связывания параметров компонента IBStoredProc и параметров процедуры (рис. 5.4). Метод ExecProc выполняет процедуру.

Создадим новое приложение. На форму поместим компоненты IBDataBase, IBTransaction. Для вывода таблицы SPECIALISTS на форму помещаем компоненты IBTable1, DataSource1, DBGrid1.

Для вывода таблицы ACTIONS_GROUPS на форму помещаем компоненты IBTable2, DataSourse2, DBGrid2.

На форму помещаем кнопку для вызова исполняемой процедуры (рис. 5.3).

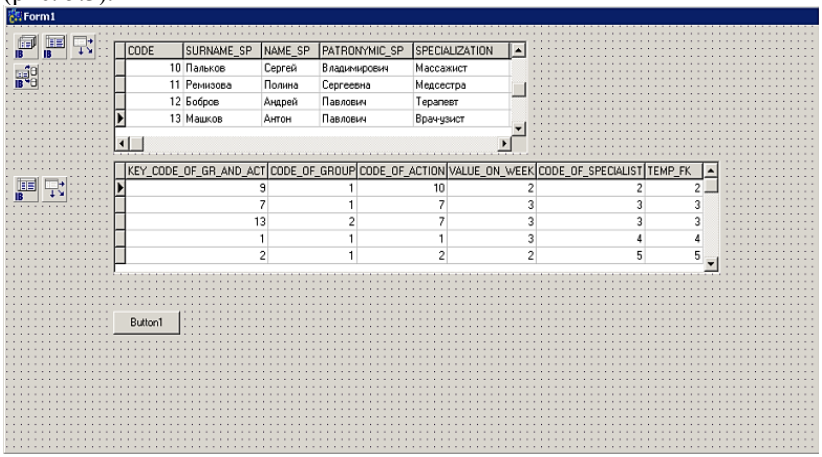


Рис 5.3. Заготовка формы с двумя таблицами

Связываем таблицы (рис.5.4) :

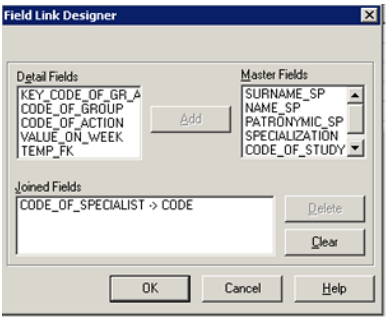


Рис.5.4.Окно формирования свойства MasterFields

Код исполняемой процедуры DELETESPEC:

```
DELETE FROM  actions_groups
where CODE_OF_specialist = :idSpec;
DELETE FROM specialists
WHERE specialists.code = : idSpec;
suspend;
END
```

Для запуска исполняемой процедуры используем компонент IBStoredProc (рис. 5.5):

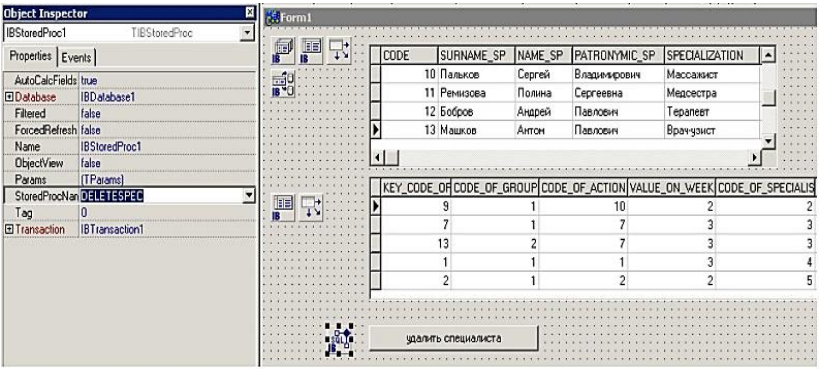


Рис. 5.5. Форма, реализующая вызов исполняемой процедуры DELETESPEC

Код обработчика события нажатия кнопки **“**Удалить специалиста”:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int a;
a = IBTable1->FieldByName("Code")->AsInteger;    // в переменную
a заносим текущее
                                                  // значение поля
“CODE”
// инициализируем входной параметр "IDSPEC" значением
переменной a
```

IBStoredProc1->ParamByName("IDSPEC")->Value = a;

IBStoredProc1->ExecProc(); // выполняем хранимую процедуру.

IBTable1->Refresh();
IBTable2->Refresh();
}

Для примера удалим специалиста с кодом 3 (рис.5.6, 5.7).

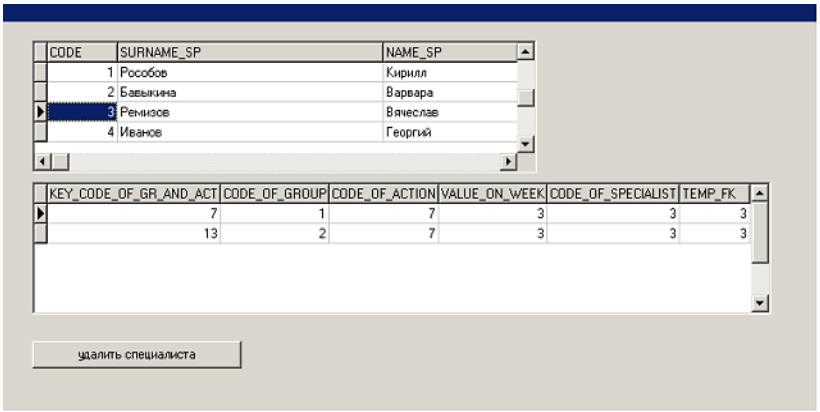


Рис. 5.6. Форма с выбранной записью для удаления

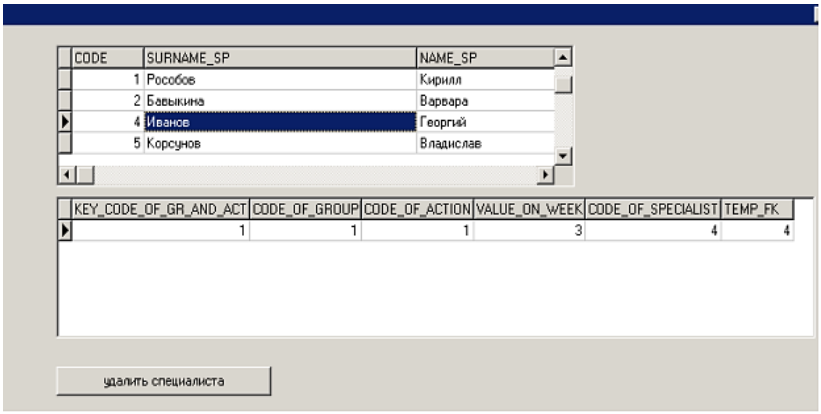


Рис 5.7. Форма после удаления специалиста с кодом 3

Задание

Для улучшения работы приложения создать хранимые процедуры.

Лабораторная работа № 6

Составление отчетов

Для создания отчетов в C++Builder включена система QuickReport. Компоненты этой системы размещены на странице QReport палитры компонентов.

QuickReport использует генератор отчетов, состоящих из множества полос. Полоса (band) – это область отчета или раздел, содержащий некоторый текст, изображения, графики, диаграммы и т.п. Полоса является контейнером для других компонентов, вносящих в отчет информацию или графику.

Если полоса и размещенные на ней компоненты связаны с базой данных, то содержание этой полосы печатается столько раз, сколько соответствующих записей имеется в источнике данных. Таким образом, достаточно расположить компоненты, связанные с данными, на полосе, а печатаемые значения и их количество будут автоматически управляться базой данных.

На практике компонент QuickRep обычно связывается с набором данных, записи которого отображаются на форме в визуальных компонентах. В этом случае в отчет попадают записи, удовлетворяющие, например, критерию фильтрации.

Пусть у нас есть форма, на которой выводится информация об отдыхающих санатория. Причем при выводе используется фильтр по заболеванию (рис.6.1).

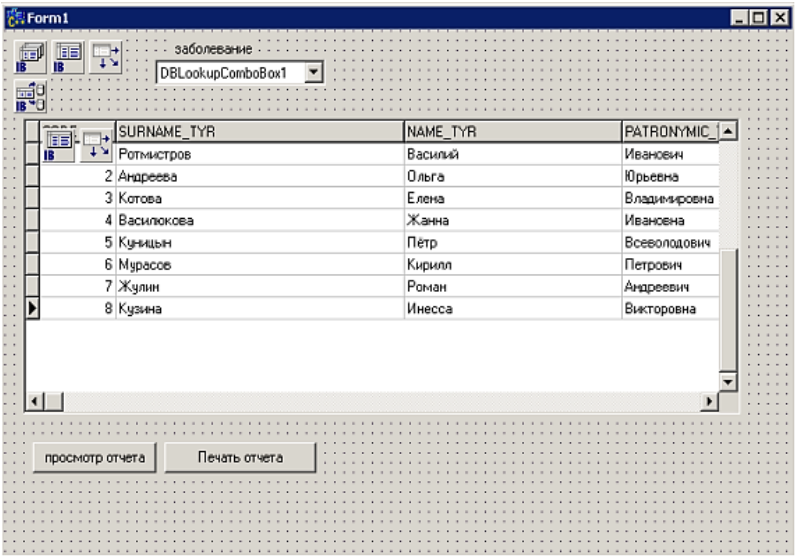


Рис. 6.1. Форма, выводящая информацию об отдыхающих,
с фильтром по заболеванию

Разместим на форме кнопки просмотра и печати отчета, обработчики которых включают в себя следующий код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Form2->QRLabel7->Caption = DBLookupComboBox1->Text;
 Form2->QuickRep1->Preview();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 Form2->QRLabel7->Caption = DBLookupComboBox1->Text;
 Form2->QuickRep1->Print();
}
```

Проектировать отчет будем на форме Form2 (рис. 6.2).

Отчет состоит из отдельных полос – составных частей отчета. Полоса предназначена для отображения соответствующих компонентов отчета и вывода данных.

Управлять наличием полос в отчете можно с помощью свойства Bands.

Добавим в отчет полосу HasTitle – заголовок отчета и HasPageFooter – нижний колонтитул.

В полосе заголовка и разместим компоненты QRLabel  QRSysData 

В полосе нижнего колонтитула также разместим эти компоненты.

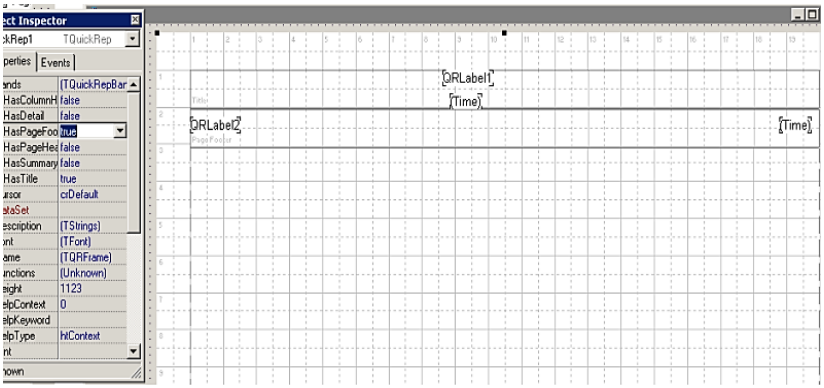


Рис.6.2. Форма простейшего отчета.

Для компонента QRSysData в заголовке свойству **Data** зададим значение qrsData, что позволит выводить с помощью этого компонента декущую дату (рис.6.3).

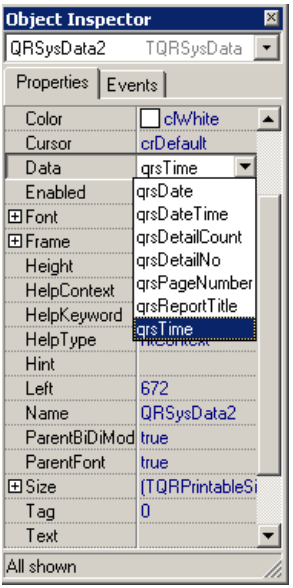


Рис. 6.3. Окно инспектора объектов с указанием значения qrsData свойству Data компонента QRSysData

В нижнем колонтитуле для компонента QRSysData свойству **Data** зададим значение qrsPageNumber для вывода номера текущей страницы отчета (рис.6.4).

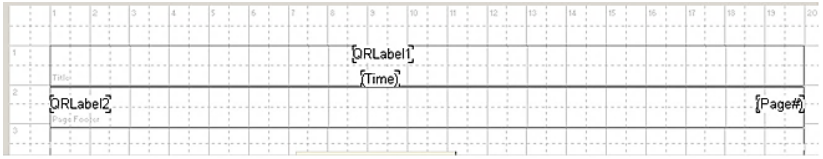


Рис. 6.4. Форма с указанием значения qrsPageNumber свойству Data компонента QRSysData

Компонент QRSysData позволяет перед системной информацией добавлять текстовую строку с помощью свойства Text. Заполним это свойство, а также свойство Caption компонентов QRLabel как показано на рис. 6.5:

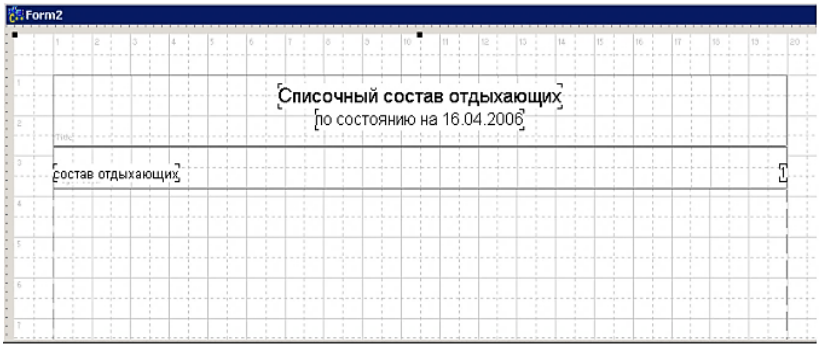


Рис.6.5. Форма с указанием свойств Text и Caption

Компонент QuickRep связывается с набором данных Table или Query с помощью свойства DataSet.

Отчет QuickRep1 находится на форме Form2 (рис.6.6), поэтому в файле реализации модуля формы Form2 следует поместить директиву препроцессора #include “unit1.h” для подключения заголовочного файла модуля формы Form1.

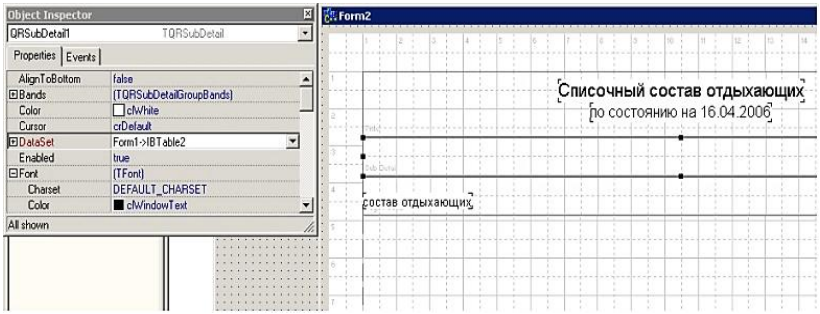


Рис. 6.6. Подключение заголовочного файла модуля формы Form1

Для вывода заголовков столбцов таблицы в его свойстве Bands включите опцию HasColumnHeader, а для вывода самих данных включите опцию HasDetail (рис.6.7).

На полосе Detail разместите компоненты QRDBText, которые необходимо связать с полями наборов данных (задать значения свойствам DataSet и DataField) (рис. 6.8).

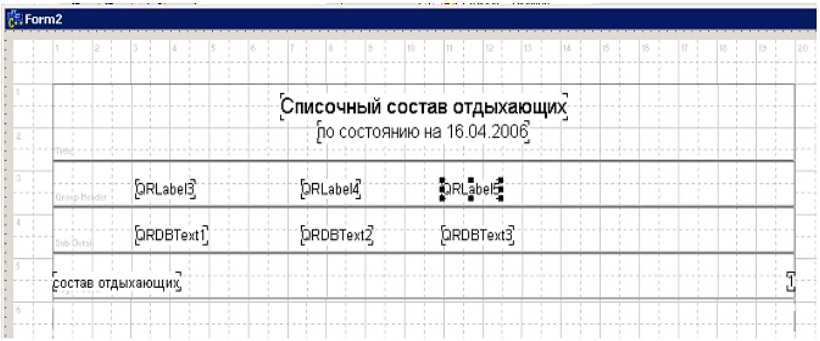


Рис. 6.7. Форма, с указанием заголовков столбцов

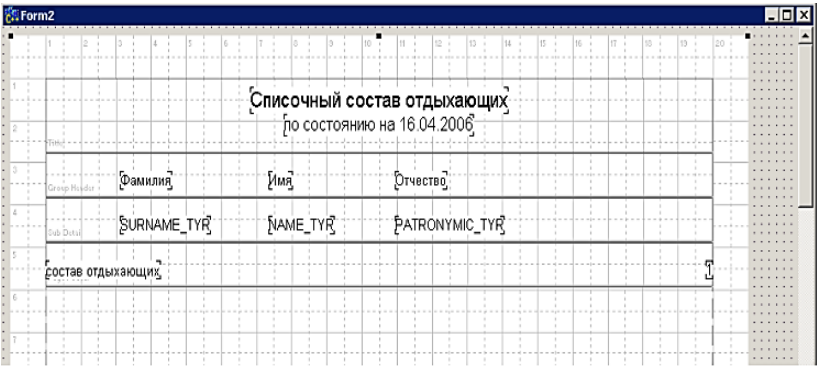


Рис. 6.8. Форма, с указанием названия столбцов

Отчет, созданный с помощью системы Quick Report с данными из одной таблицы

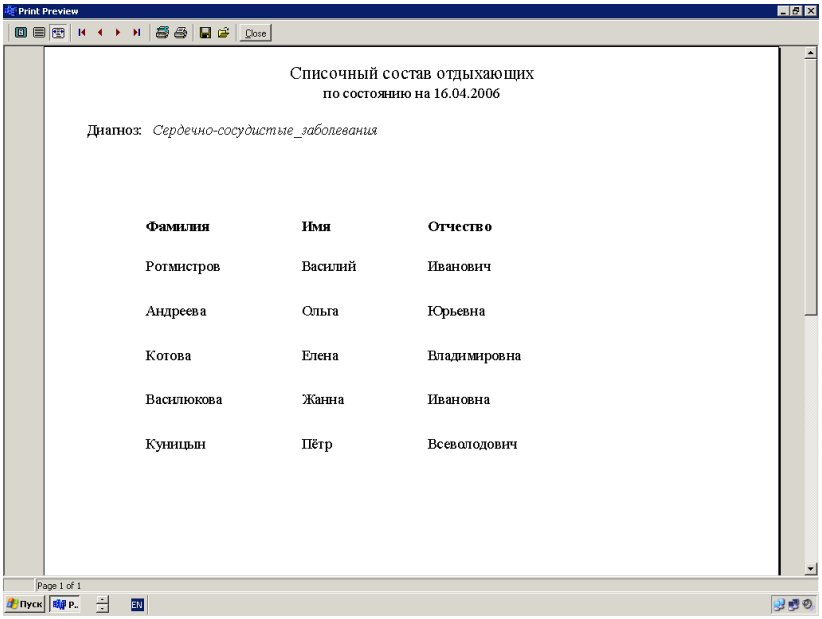


Рис. 6.9. Отчет, созданный с помощью системы Quick Report с данными из одной таблицы

Для вывода данных из связанных по внешнему ключу таблиц нужно использовать компонент QRSubDetail, который связывается с дочерней таблицей.

Например, нужно включить в отчет всех отдыхающих и сгруппировать их по заболеванию (рис. 6.9).

Создадим новый отчет. Добавим на него два компонента QRSubDetail .

Соединим первую полосу с компонентом IBTable1, а вторую с IBTable2 (заполнить свойство DataSet). На полосы разместим компоненты QRDBText и свяжем их с соответствующими полями.

Для добавления заголовков столбцов дочерней таблицы включим в отчет дополнительную полосу (компонент QRBand) . Для этой полосы зададим свойству BandType значение GroupHeader. В компоненте QRSubDetail, связанному с дочерней таблицей, в свойстве

HeaderBand укажем на добавленную полосу. После этого размещаем на данной полосе метки. В результате получим следующую структуру отчета (рис. 6.10) и сам отчет (рис. 6.11):

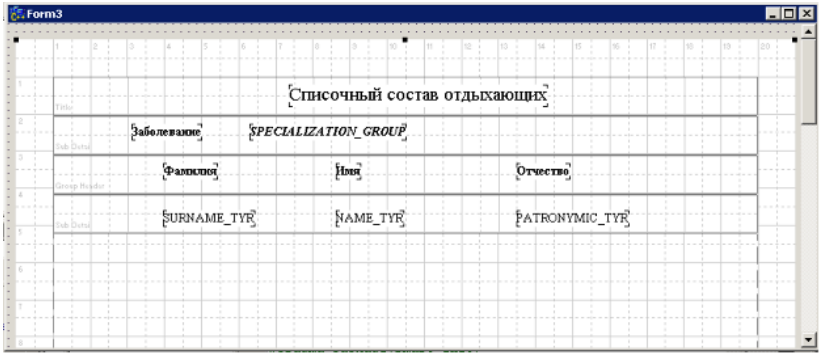


Рис. 6.10. Структура отчета Списочный состав отдыхающих

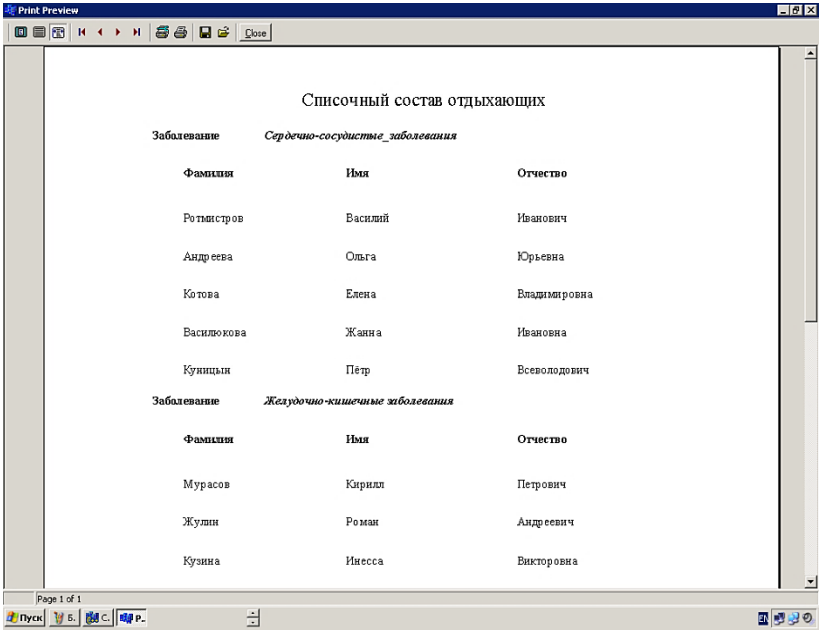


Рис. 6.11. Списочный состав отдыхающих

Чтобы подключить к форме приложения отчет с вычислением количества отдыхающих, выполните следующие действия:

1. Включите ещё один компонент QRBand в лист отчета. Свойство BandType определяет роль rbSummary данной полосы – отображать итоговые вычисления.

2. Пометите компонент QRExpress на полосе Summary. Откройте редактор свойств Expression и скомпилируйте формулы для подсчета числа служащих и вычисления количества отдыхающих COUNT.

3. В свойстве Master нужно указать полосу, в которую выводится информация об отдыхающих (QRSubDetail2).

4. Скомпилируйте приложение.

Структура отчета показана на рис. 6.12. Рис. 6.13. показывает отчет с вычислением количества отдыхающих на этапе проектирования.

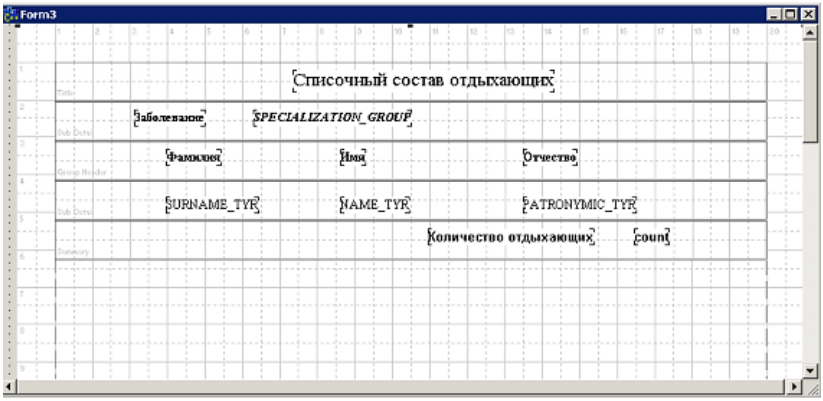


Рис. 6.12. Структура отчета с вычислением количества отдыхающих

На рис.6.13 показан отчет, полученный в результате выполнения, проектируемого отчета с использованием статистических функций.

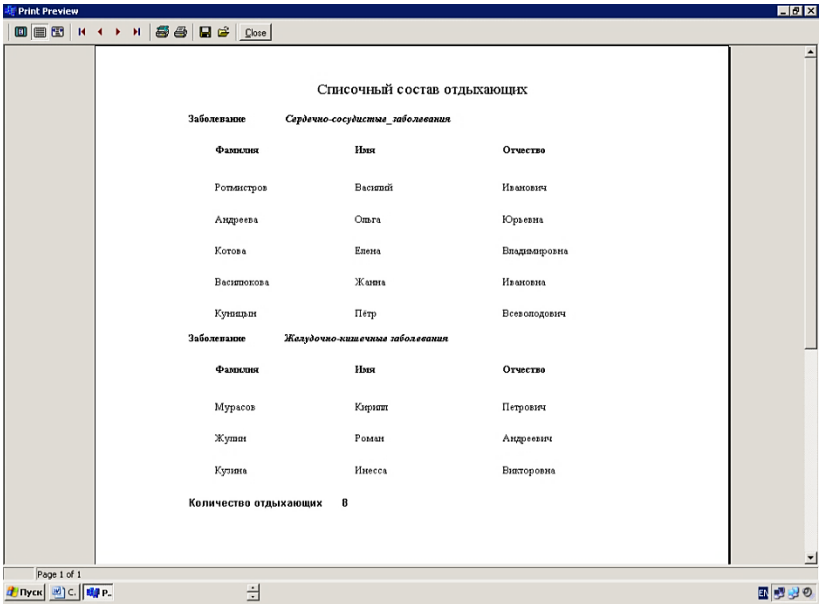


Рис. 6.13. Отчет с использованием статистических функций

Создание отчета в Word. При составлении отчетов, содержащих сведения, черпаемые из баз данных, можно использовать компоненты системы **QuickReport**. Однако **QuickReport** накладывает на форму отчетов достаточно жесткие ограничения. На основе **QuickReport** можно разрабатывать приложения для некоторых стандартных отчетов с часто обновляемыми данными. Но нередко хотелось бы иметь значительно большую свободу при компоновке и написании отчетов, хотелось бы иметь возможность вставлять данные в некий произвольный текст и в произвольной форме. Такую свободу дает программа **Windows Word**, широко используемая каждым, имеющим дело с персональными компьютерами.

Создадим отчет на основе следующего шаблона:

Списочный состав отдыхающих
по состоянию на 06.10.2015

Диагноз:

№	Фамилия	Имя	Отчество
1			

На форму приложения добавим компоненты **WordApplication** и **WordDocument**.

При использовании свойства **AutoConnect** надо иметь в виду, что установка в **true** влияет только при запуске приложения, т.е. если это свойство установлено во время проектирования. Задание **AutoConnect=true** во время выполнения приложения ни на что не влияет.

Свойство **ConnectKind** определяет, как именно осуществляется соединение с сервером. Это свойство может принимать следующее значение:

ckRunningOrNew	Подсоединиться к выполняющемуся серверу или создать новый экземпляр сервера.

Свойства и методы сервера Word

Описание сервера **Word,** как правило, можно найти во встроенной справке **Word**. Изложение в справке ведется на основе языка **Visual Basic**.

Обращение к свойствам объекта **WordApplication**, инкапсулирующего объект **Application** (этот объект является самым выполняющимся экземпляром **Word**), производится так же, как в свойствах любого объекта **C++Builder**. Например, в **Application** имеется свойство **Options** – опции, являющейся в свою очередь объектом со множеством свойств. Среди этих свойств есть **CheckSpelligAsYouType** и **CheckGrammarAsYouType** – булевы свойства, указывающие, должен ли **Word** автоматически проверять синтаксис и грамматику и отмечать в тексте ошибки. Такая проверка замедляет работу **Word.** Если вы хотите отключить в сервере эти автоматические проверки, следует ввести в приложение операторы:

WordApplication→ Options→ CheckSpelligAsYouType = false;
WordApplication→ Options→ CheckGrammarAsYouType = false;

Тем самым вы отключите автоматические проверки, тем более что в случае, если **Word** невидим и работает за кадром, эти проверки совершенно бессмыслены.

Среди множества свойств **Application** следует отметить свойство **ActiveDocument** – активный документ. Это объект **Document**, некоторые свойства и методы которого будут описаны ниже.

Практически всегда при работе с сервером **Word** вам приходится иметь дело со свойством **Documents**. Это свойство представляет собой собрание всех документов, открытых в **Word** в данный момент. Каждый документ представлен в этом собрании как объект **Document**, имеющий в свою очередь собственные свойства и методы. Общее число открытых документов определяется свойством **Count** собрания документов **Documents**. Это свойство только для чтения часто приходится проверять, чтобы узнать, если в **Word** хотя бы один открытый документ. Например, если в вашем приложении предусмотрены действия **ASave**, **APrint** и **APreview**, обеспечивающие сохранение, печать и предварительный просмотр документа, то их, очевидно, надо делать недоступными, если ни одного документа в **Word** нет. Это можно осуществить следующим кодом:

```
If ( WordApplication→ Documents→ Count = 0)
 {
    ASave→Enabled = false;
    APreview→ Enabled = false;
    APrint→ Enabled = false;

    }
```

Создание нового документа **Document** и включение его в **Documents** осуществляется методом **Add** объекта **Documents**. В этот метод можно передать несколько необязательных аргументов. Их число зависит от используемой версии компонента **WordApplication**. Первые два аргумента в обеих версиях одинаковы: **Template** и **NewTemplate**. Аргумент **Template** указывает шаблон, который используется при создании документа. Если этот аргумент не указан, то документ создается на основе шаблона **Обычный(Normal)**. Аргумент **NewTemplate** булева типа определяет, открывается ли документ как шаблон (при значении **true**), или как обычный документ. По умолчанию **NewTemplate – false**, т.е. открывается обычный документ. В **WordApplication** все ограничивается этими двумя документами.

При вызове из **C++Builder** любого метода сервера **COM** аргументы (кроме аргументов типа **Text**) передаются только как объекты типа **OleVariant**. Если какие-то аргументы не являются обязательными, то все равно они должны фигурировать в вызове метода. Только вместо их значений может быть указана **EmptyParam** – переменная типа **OleVariant**, используемая вместо необязательных параметров. Эта переменная объявлена в модуле **System.**

Таким образом, для **WordApplication** , если вы хотите создать новый документ на основе обычного шаблона, вы можете записать оператор:

WordApplication1→ Documents→ Add(EmptyParam, EmptyParam);

Но если нужно создать документ на основе своего шаблона, например, **C:\MyTemplate\My.dot**, то код будет сложнее:

TVariant Template = "C:\\MyTemplate\\My.dot";

WordApplication1→ Documents→ Add(&Template, EmptyPara

А если требуется создать документ как новый шаблон на основе обычного шаблона, то код будет следующим:

TVariant Template = true;

WordApplication1→ Documents→ Add(EmptyParam, &Template);

При передачи булевых аргументов можно использовать значение 0 вместо false и целое ненулевое значение (например,1) вместо true. Поэтому, в последнем варианте, когда задание значения NewTemplate можно выполнить следующим оператором:

TVariant Template = 1;

Таким образом, при передаче булевых аргументов и свойств можно использовать две различных формы записи значения.

Важным свойством сервера **Word** является свойство **Selection**, являющееся ссылкой на объект **Selection** – выделенный фрагмент текста в активном документе или, если нет выделения, то просто текущая позиция курсора в активном документе. Этот объект имеет методы **InsertBefore** и **InsertAfter**, аргументом в которые передается текст, вставляемый в активный документ соответственно до или после объекта **Selection**. Например, код:

WordApplication1→ Selection→ InsertAfter(TVariant('\n'));

WordApplication1→ Selection→ InsertAfter(TVariant("Дорогой"+Edit1→Text+"!\n"));

Вставляет после **Selection** пустую строку, а затем вставляет строку с текстом "Списочный ….", где вместо многоточия фигурирует текст окна редактирования **Edit1**. Если выделения текста не было, то

все это **WordApplication1**вставляется после текущей позиции курсора. В результате курсор перемещается на первую позицию после вставленного текста, а весь вставленный текст выделяется, т.е. включается в **Selection.**

Выше рассматривалась вставка текста с помощью методов **InsertBefore** и **InsertAfter**. Кроме этих методов имеется еще один метод вставки текста **TypeText**. В него, так же, как и в методы **InsertBefore** и **InsertAfter**, передается один аргумент – текст. Выполнение метода **TypeText** зависит от значения свойства **ReplaseSelection** объекта **Options**. Если оно установлено в **true**, то все выделение заменяется текстом, переданным как аргумент. Если **ReplaseSelection – false**, то новый текст вставляется перед выделением. По умолчанию **ReplaseSelection – true.** Таким образом операторы

WordApplication1→ Options→ ReplaseSelection = true;

WordApplication1→Selection→TypeText(TVariant("новый текс"));

приведут к тому, что выделенный текст замениться словами "новый текст". Первый из приведенных операторов можно не писать, если есть уверенность, что значение **ReplaseSelection,** равное true по умолчанию, не изменено какими то предыдущими операторами.

Вставку в документ нового текста или изображения можно осуществлять также методом **Paste**, который копирует информацию из буфера обмена **Clipboard**. Например, операторы:

```
#include< Clipboard.hpp>
……..
Clipboard()→Assign(Image1→Picture);
WordApplication1→Selection→Paste();
```

Осуществляет копирование с помощью буфера обмена в текущую позицию курсора в документ изображение, хранящееся в компоненте **Image1**. Выполнение метода Paste так же, как и метода **TypeText**, зависит от значения деления, или вводиться перед ним.

В заключении остановимся еще на одном свойстве сервера **Word – Dialog**. Это коллекция объектов **Dialog**, который соответствует встроенным диалогам **Word.** Доступ к конкретному диалогу осуществляется через выражение вида

WordApplication1→ Dialogs→Item(WdWord Dialog),

где константа **WdWordDialog** может принимать одно из предопределенных значений. В приведенной ниже таблице даются некоторые из этих значений и описание диалогов, которым они соответствуют.

wdDialogEditFind	Найти фрагмент текста
wdDialogEditPasteSpecial	Специальная вставка из буфера обмена
wdDialogEditReplace	Заменить фрагмент текста
wdDialogFileFind	Найти файл
wdDialogFileNew	Новый файл
wdDialogFileOpen	Открыть файл
wdDialogFilePageSetup	Параметры страницы
wdDialogFilePrint	Печать файла
wdDialogFilePrintSetup	Установить принтер
wdDialogFileSaveAs	Сохранить файл как
wdDialogFileSummaryInfo	Свойства документа (Статистика)
wdDialogFormatFont	Шрифт
wdDialogFormatParagraph	Абзац
wdDialogInsertDatabase	Вставить базу данных
wdDialogInsertFile	Вставить файл
wdDialogInsertPageNumbers	Вставить номера страниц

Из методов, которые имеют объекты **Dialog**, остановимся только на одном – методе **Show**. Он открывает пользователю соответствующее диалоговое окно и выполняет те команды, которые указал в нем пользователь.

Печать документа без отображения диалога печати может осуществляться методом **PrintOut:**

WordDocument1→PrintOut();

Предварительный просмотр документа перед печатью осуществляется методом **PrintPreview:**

WordDocument1→ PrintPreview();

На рис. 6.14 представлена форма на этапе проектирования, позволяющая осуществить предварительный просмотр отчета, перед его печатью. На рис. 6.15 отчет представлен в **Word**.

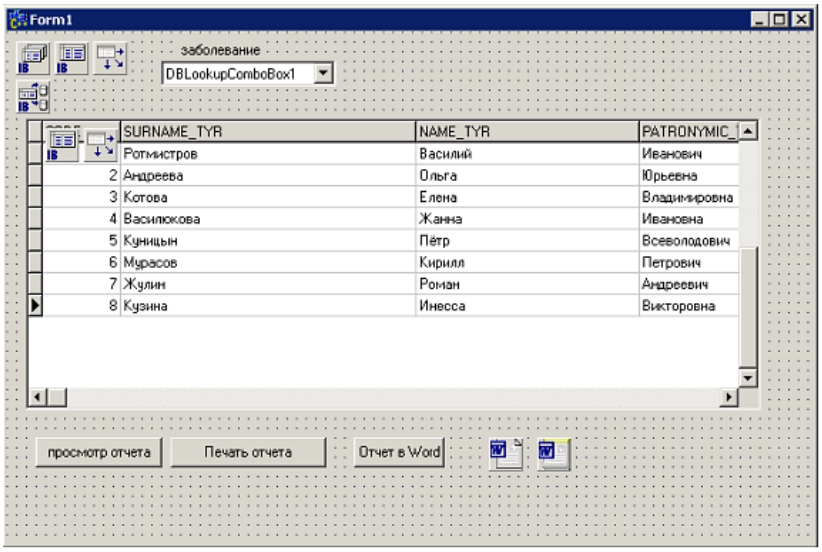


Рис. 6.14. Форма для просмотра и печати отчета в Word

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
int i;
TVariant Template;
Template = "E:\\student\\ИТ-21\\15.05.15\\список.dot";
 WordApplication1->set_Visible(1);  // окно Word делаем видимым
// при добавлении нового документа используем шаблон список.dot
 WordApplication1->Documents->Add(Template, EmptyParam);
// соединяем компонент WordDocument1 с активным документом.
 WordDocument1->ConnectTo(WordApplication1->ActiveDocument);
// В пятый абзац добавляем информацию из IBTable1
WordDocument1->Paragraphs->Item(5)->Range-
>InsertBefore(TVariant(IBTable1->Fields->Fields[2]->AsString));
// перемещаем курсор в начало таблицы “Отдыхающие”
IBTable2->First();
i = 2;
```

```
// Перебираем записи таблицы “Отдыхающие” и записываем их в докумет.
while(!IBTable2->Eof)
{
WordDocument1->Tables->Item(1)->Cell(i,2)->Select();
WordApplication1->Selection->InsertAfter(TVariant(IBTable2->FieldByName("SURNAME_TYR")->AsString));
WordDocument1->Tables->Item(1)->Cell(i,3)->Select();
WordApplication1->Selection->InsertAfter(TVariant(IBTable2->FieldByName("NAME_TYR")->AsString));
WordDocument1->Tables->Item(1)->Cell(i,4)->Select();
WordApplication1->Selection->InsertAfter(TVariant(IBTable2->FieldByName("PATRONYMIC_TYR")->AsString));
// добавляем новую строку в таблицу документа
WordDocument1->Tables->Item(1)->Rows->Add();  i++;
//перемещаем курсор на одну строку вниз.
}
Sub ll()
Dim RetVal
RetVal = Shell("E:\student\Project2.exe", 1)
End Sub          }
```

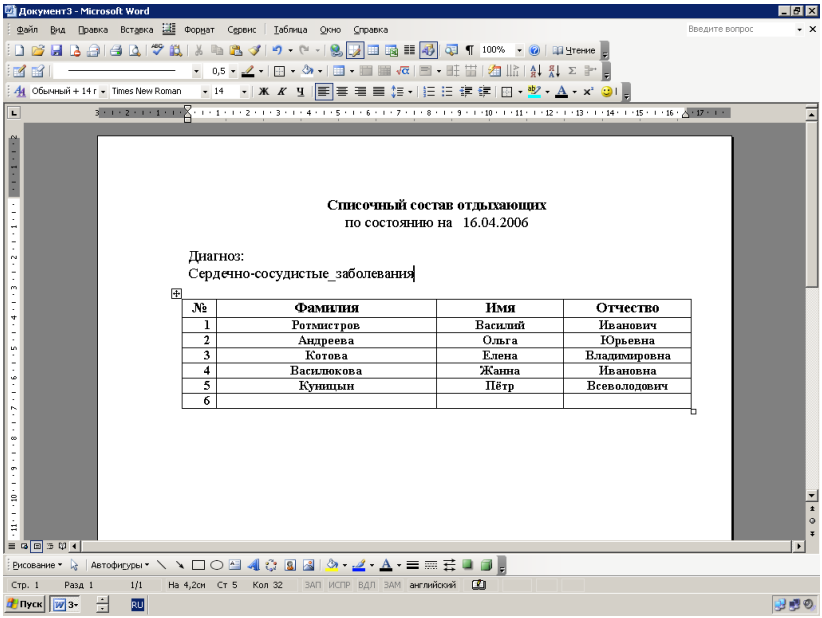


Рис. 6.15. Отчет конвертируемый в Word

Задание

Для создаваемого приложения создать отчеты.

Библиографический список

1. Культин Н. Б. C++Builder в задачах и примерах. – СПб.:БХВ-Петербург, 2005.- 336 с.

2. Архангельский А. Я. Программирование в C++Builder6.- М.:БИНОМ, 2003.- 1152 с.

3. Хомоненко А., Ададуров С. Работа с базами данных в C++Builder. СПб.:БХВ-Петербург, 2006.- 1438 с.

Учебное издание

Чернова Светлана Борисовна
Шаптала Вадим Владимирович

Информационные технологии

Лабораторный практикум
Часть 2

Подписано в печать Формат 60×84/16. Усл. печ. л. 4,5. Уч.-изд. л. 4,9.

Тираж 83 экз. Заказ Цена

Отпечатано в Белгородском государственном технологическом университете
им. В.Г. Шухова
308012, г. Белгород, ул. Костюкова, 46.