МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ
«ЛУГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ ВЛАДИМИРА ДАЛЯ»

Стахановский инженерно-педагогический институт менеджмента
Кафедра информационных систем

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

«УПРАВЛЕНИЕ ИНФОРМАЦИЕЙ И ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ»

для студентов направления подготовки
Профессиональное обучение (по отраслям),
профиль «Информационные технологии и системы»
(в 2-х частях). Часть 1.

Луганск 2023

УДК 004.912, 004.62, 004.63

Рекомендовано к изданию Учебно-методическим советом
ГОУ ВО ЛНР «ЛГУ им. В. ДАЛЯ»
(протокол № от . .2023 г.)

Методические указания к лабораторным работам по дисциплине «**Управление информацией и интеллектуальные системы**» для студентов направления подготовки **Профессиональное обучение (по отраслям),** профиль **«Информационные технологии и системы»** (в 2-х частях). Часть 1. / Сост.: Е.С. Чёрная. – **Стаханов:** ГОУ ВО ЛНР «ЛГУ им. В. Даля», 2023. – 46 с.

Методические указания к лабораторным работам определяют планирование, организацию и проведение лабораторных работ по учебной дисциплине «Управление информацией и интеллектуальные системы». Лабораторные работы относятся к основным видам учебных занятий и составляют важную часть теоретической и профессиональной практической подготовки. Представленные лабораторные работы направлены на формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности. В методических указаниях представлены восемь лабораторных работ. В методических рекомендациях в лаконичной форме дана справочная информация по изучаемой теме, методические указания студентам по выполнению лабораторных работ, эталон решения задачи, контрольные задания в необходимом количестве вариантов, дающие возможность обеспечить индивидуальное выполнение задания студентом

Предназначены для студентов профиля «Информационные технологии и системы».

Составитель: доц. Чёрная Е.С.

Ответственный за выпуск: доц. Карчевский В.П.

Рецензент: доц. Карчевская Н.В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Управление информацией и интеллектуальные системы» входит в вариативную часть профессионального цикла дисциплин подготовки студентов по направлению подготовки 44.03.04 Профессиональное обучение (по отраслям).

Дисциплина реализуется кафедрой информационных систем.

Основывается на базе дисциплин: «Информатика и информационные технологии», «Производственное обучение», «Программное обеспечение систем управления и обучения».

Является основой для изучения следующих дисциплин: «Программная инженерия», «Объектно-ориентированное программирование и технологии разработки программного обеспечения».

Целью изучения дисциплины «Управление информацией и интеллектуальные системы» является знакомство с разнообразными информационными технологиями управления базами данных информационных систем; рассмотрение основных подходов к проектированию информационных систем, концепции баз данных; получение студентами глубоких и систематизированных знаний о методологии проектирования баз данных, о проектировании автоматизированных систем управления предприятием.

Основной **задачей** изучения дисциплины «Управление информацией и интеллектуальные системы» является: получение студентами необходимых теоретические и практических знаний работы с различными информационными технологиями управления базами данных информационных систем.

Студенты, завершившие изучение дисциплины «Управление информацией и интеллектуальные системы», должны

знать:

- определение интеллектуальных систем, структуру статических и динамических экспертных систем;
- теоретические основы построения и функционирования прикладных информационных систем, ключевые направления применения новых информационных систем при автоматизации процессов принятия управленческих решений;
- методы построения эксплуатации и разработки информационных систем;
- модели представления знаний;
- возможности интеллектуальных систем и имеющихся программных продуктов;
- основные источники научно-технической информации по основным направлениям, методам, моделям и инструментальным средствам конструирования информационных систем;

уметь:

- формулировать цели и задачи автоматизации обработки управленческой информации;
- применять интеллектуальные системы для решения задач оценки и прогнозирования состояния объектов;
- оперировать с объектами баз данных - таблицами, формами и запросами, отчетами, макросами;
- управлять базой данных с помощью различных элементов управления;
- создавать сложные формы с использованием различным элементов управления;

настраивать интерфейс пользователя базы данных;

***владеть навыками*:**

- терминологией, навыками поиска и использования научно-технической информации по профессиональной тематике;
- разработки концепции интеллектуальных систем;
- анализа информационных систем управления персоналом.

Перечисленные результаты образования являются основой для формирования следующих компетенций (*в соответствии с государственными образовательными стандартами ВО и требованиями к результатам освоения основной образовательной программы (ООП):*

общекультурных:

ОК-3	-	способность использовать основы естественнонаучных и экономических знаний при оценке эффективности результатов деятельности в различных сферах
ОК-6	-	способность к самоорганизации и самообразованию

общепрофессиональных:

ОПК-6	-	способность к когнитивной деятельности
ОПК-10	-	владение системой эвристических методов и приемов

профессиональных:

ПК-1	-	способностью выполнять профессионально-педагогические функции для обеспечения эффективной организации и управления педагогическим процессом подготовки рабочих, служащих и специалистов среднего звена
ПК-9		готовность к формированию у обучающихся способности к профессиональному самовоспитанию.
ПК-13	-	готовностью к поиску, созданию, распространению, применению новшеств и творчества в образовательном и технико-технологическом процессах для решения профессионально-педагогических и производственно-технологических задач

ЛАБОРАТОРНАЯ РАБОТА 1

СОЗДАНИЕ ТАБЛИЦ В DATABASE DESKTOP В DELPHI.

Цель работы:

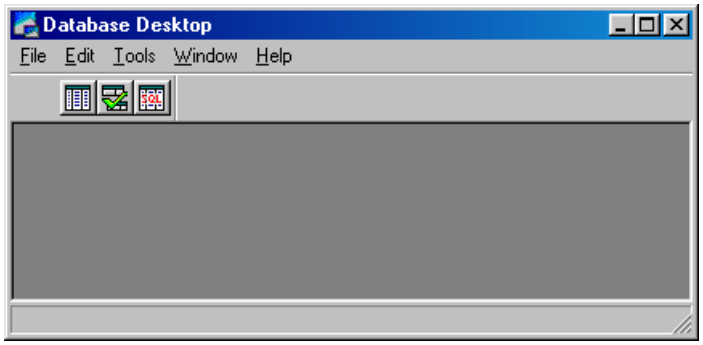
1. Ознакомить с утилитой Database Desktop.
2. Получить навыки создания и редактирования таблиц с помощью Database Desktop.

Database Desktop - это утилита, которая поставляется вместе с Delphi для интерактивной работы с таблицами различных форматов локальных баз данных - Paradox и dBase, а также SQL-серверных баз данных InterBase, Oracle, Informix, Sybase (с использованием SQL Links). Исполняемый файл утилиты называется **DBD32.EXE**. Для запуска Database Desktop просто дважды щелкните по ее иконке.

Запуск Database Desktop:

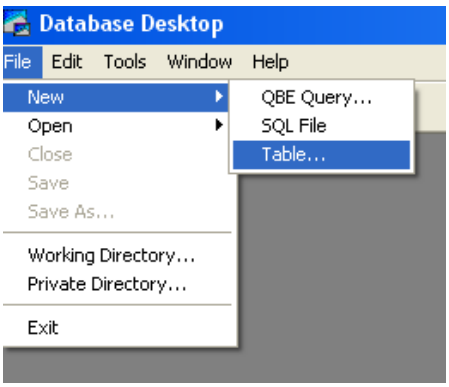
1. Запустить Delphi
   1.1 В меню Delphi выбрать раздел **Tools**
   2.2 В появившемся списке выбрать строку **Database Desktop**
2. Щелкнуть по кнопке **Пуск**
   1.1 Из главного меню выбрать строку **Программы**
   2.2 В появившемся списке выбрать строку **Delphi 7**
   3.3 В следующем списке выбрать строку **Database Desktop**

После запуска Database Desktop на экране появится окно:

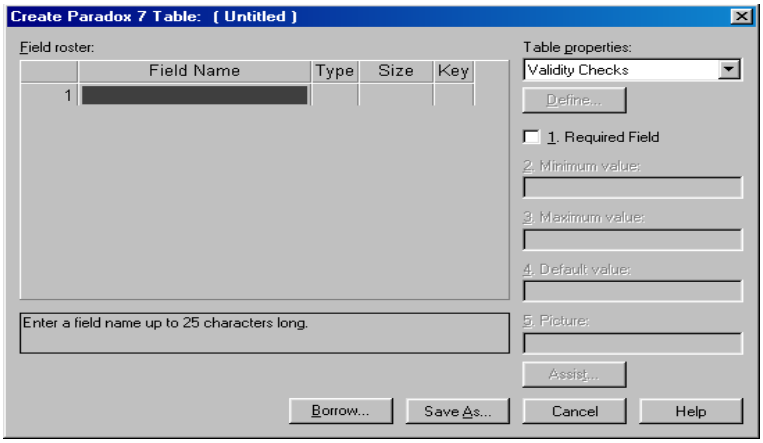


Создание таблиц в Database Desktop:

1. Запускаем программу Database Desktop;
2. В появившемся окне выбираем: **File►New►Table.**



Формат таблицы выбираем **Paradox.** После этого появится окно создания таблицы, в котором можно определить поля таблицы и их тип:



По умолчанию сразу после открытия окна в правой его части в списке **Table properties** выбран пункт **Validity Chekcs**, что позволяет контролировать содержимое полей. С помощью флажка **Required Fields** можно потребовать обязательного заполнения поля при вводе новой записи. Также можно контролировать минимальное и максимальное значение числового поля в строках **Minimum Value** и **Maximum Value**. В строке **Default Value** можно указать значение поля по умолчанию – при вводе новой записи значение в это поле поместит **BDE**.С помощью строки **Picture** можно задать шаблон для автоматического форматирования значения поля. Например, если задан шаблон (###)###-#### и в поле введена строка 9054005647, она будет автоматически преобразована к виду (905)400-5647.

Предназначение кнопок:

Borrow… – осуществляет копирование структуры таблицы из другой таблицы.

Save as… – сохраняет изменения в структуре таблицы.

Cancel – выход без сохранения.

Help – вызов справки.

3. Чтобы определить структуру таблицы в этом окне необходимо заполнить следующие графы:
   - **Field Name - Имя поля.**
   - **Type - Тип поля.** Вызывает список допустимых типов, щелчком правой кнопки мыши или клавишей пробел.
   - **Size - Размер.** Определяет размер поля. Не все типы полей имеют размер. Большинство типов имеют стандартный размер, который не может быть изменен. Размер в основном меняется у строковых типов (Alpha), бинарных (Binary) и др.
   - **Key - Ключ**. Двойной щелчок мышью определяет ключевое поле. Ключевыми могут быть только первые поля, то есть второе поле сможет быть ключевым только вместе с первым.

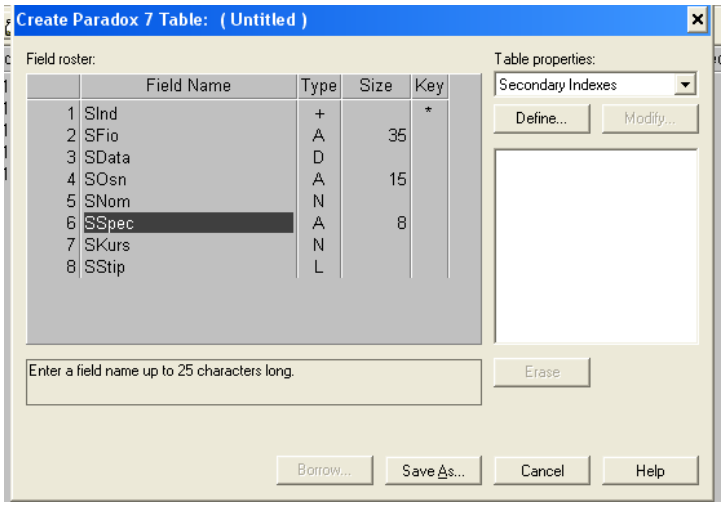
Типы полей формата Paradox

Alpha	строка длиной 1-255 байт, содержащая любые печатаемые символы
Number	числовое поле длиной 8 байт, значение которого может быть положительным и отрицательным. Диапазон чисел - от 10^{-308} до 10^{308} с 15 значащими цифрами
$ (Money)	числовое поле, значение которого может быть положительным и отрицательным. По умолчанию, является форматированным для отображения десятичной точки и денежного знака
Short	числовое поле длиной 2 байта, которое может содержать только целые числа в диапазоне от -32768 до 32767
Long Integer	числовое поле длиной 4 байта, которое может содержать целые числа в диапазоне от -2147483648 до 2147483648
# (BCD)	числовое поле, содержащее данные в формате BCD (Binary Coded Decimal). Скорость вычислений немного меньше, чем в других числовых форматах, однако точность - гораздо выше. Может иметь 0-32 цифр после десятичной точки
Date	поле даты длиной 4 байта, которое может содержать дату от 1 января 9999 г. до нашей эры - до 31 декабря 9999 г. нашей эры. Корректно обрабатывает високосные года и имеет встроенный механизм проверки правильности даты
Time	поле времени длиной 4 байта, содержит время в миллисекундах от полуночи и ограничено 24 часами

@ (Timestamp)	обобщенное поле даты длиной 8 байт - содержит и дату и время
Memo	поле для хранения символов, суммарная длина которых более 255 байт. Может иметь любую длину. При этом размер, указываемый при создании таблицы, означает количество символов, сохраняемых в таблице (1-240) - остальные символы сохраняются в отдельном файле с расширением .MB
Formatted Memo	поле, аналогичное Memo, с добавлением возможности задавать шрифт текста. Также может иметь любую длину. При этом размер, указываемый при создании таблицы, означает количество символов, сохраняемых в таблице (0-240) - остальные символы сохраняются в отдельном файле с расширением .MB. Однако, Delphi в стандартной поставке не обладает возможностью работать с полями типа Formatted Memo
Graphic	поле, содержащее графическую информацию. Может иметь любую длину. Смысл размера - такой же, как и в Formatted Memo. Database Desktop “умеет” создавать поля типа Graphic, однако наполнять их можно только в приложении
OLE	поле, содержащее OLE-данные (Object Linking and Embedding) - образы, звук, видео, документы - которые для своей обработки вызывают создавшее их приложение. Может иметь любую длину. Смысл размера - такой же, как и в Formatted Memo. Database Desktop “умеет” создавать поля типа OLE, однако наполнять их можно только в приложении. Delphi “напрямую” не умеет работать с OLE-полями, но это легко обходится путем использования потоков
Logical	поле длиной 1 байт, которое может содержать только два значения - **T** (true, истина) или **F** (false, ложь). Допускаются строчные и прописные буквы
+ (Autoincrement)	поле длиной 4 байта, содержащее нередактируемое (read-only) значение типа *long integer*. Значение этого поля автоматически увеличивается (начиная с 1) с шагом 1 - это очень удобно для создания уникального идентификатора записи (физический номер записи не может служить ее идентификатором,

	поскольку в Парадоксе таковой отсутствует. В InterBase также отсутствуют физические номера записей, но отсутствует и поле Autoincrement. Его с успехом заменяет встроенная функция *Gen_id*, которую удобней всего применять в триггерах)
Binary	поле, содержащее любую двоичную информацию. Может иметь любую длину. При этом размер, указываемый при создании таблицы, означает количество символов, сохраняемых в таблице (0-240) - остальные символы сохраняются в отдельном файле с расширением .MB. Это полнейший аналог поля BLOb в InterBase
Bytes	строка цифр длиной 1-255 байт, содержащая любые данные

Создадим таблицу с данными о студентах. Укажем такие данные, как: ФИО, дата рождения, на основании какого приказа студент принят в университет, номер зачетки, специальность, курс, стипендия. Во всех текстовых полях, необходимо указать размер.



Созданную таблицу сохраняем под названием Student.db и закрываем окно создания таблиц.

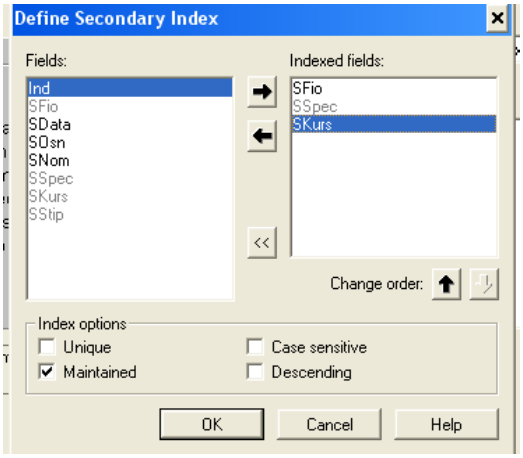
Иногда может понадобиться отредактировать уже созданную таблицу для того, чтобы добавить, изменить или удалить некоторые поля, изменить свойства таблицы.

Редактирование таблицы:

1. Определение вторичных индексов.
   1.1. Открыть таблицу Student.db (**File►Open►Table)**

1.2. Из меню **Table** выбрать пункт **Restructure**. Откроется окно редактирования полей таблицы.

1.3. В выпадающем списке **Table properties** выбрать **Secondary Indexes** и нажать кнопку **Define**. В окне **Define Secondary Index** определяются вторичные индексы:



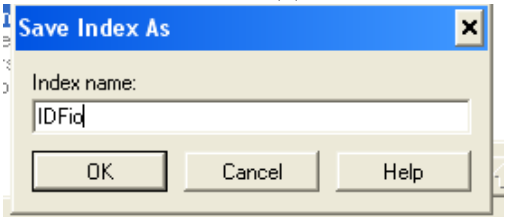
С помощью флажков группы **Index options** можно определить следующие особенности индекса:

- **Unique** – индекс будет содержать уникальные значения;
- **Maintained** – индексные поля сортируются по возрастанию значений;
- **Case sensitive** – индекс чувствителен к регистру букв в текстовых полях;
- **Descending** – индексные поля сортируются по убыванию значения.

1.4. Выбрать *«SFio»* из списка **Fields** и нажать кнопку с изображенной стрелкой вправо. В списке **Indexed fields** (индексированные поля) появится *«SFio»*. То же самое проделать с полями *«SSpec»* и *«SKurs»*.

1.5. Закрыть окно **“Define Secondary Index”**

1.6. В появившемся окне ввести имя индекса *IDFio* и нажать **"OK"**.



2. Теперь, так как на факультете всего две специальности, то можно переопределить тип поля *«SSpec»*. Для работы будет гораздо удобнее, чтоб это поле было типа **Logical**.
   2.1. Навести курсор на тип поля *«SSpec»* и написать тип поля **«L»**.
3. Определим языковой драйвер. Это следует делать для правильного отображения русскоязычного текста.
   3.1. Открыть окно редактирования полей таблицы Student.db.

3.2. В выпадающем списке **Table properties** выбрать **Table Language** и нажать кнопку **Modify**.
3.3. В появившемся окне выбрать из списка **Pdox ANSI Cyrillic**.
3.4. Сохранить таблицу.

4. Для логического типа значение по умолчанию зададим **False.**

Заполнение данными в таблице Student.db:

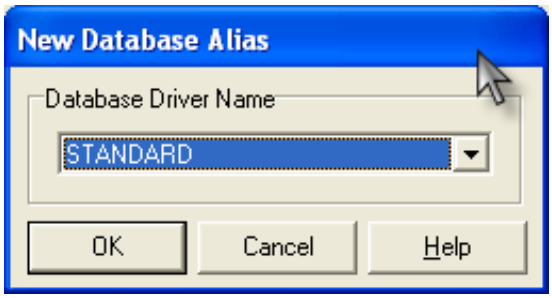
— Открыть таблицу (**File ► Open ► Table**)
— Выбрать **Table ► Edit data** или нажать клавишу **F9**.
— Создать несколько записей.

Создание псевдонима:

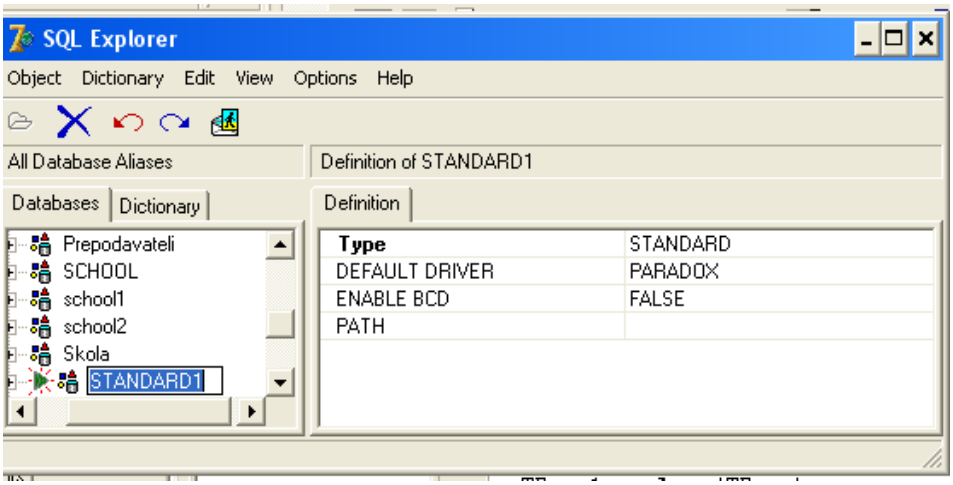
Псевдоним указывает местонахождение файлов БД и представляет собой специальное имя для обозначения каталога. Использование псевдонимов существенно облегчает перенос файлов БД в другие каталоги и на другие компьютеры. При этом не требуется изменять приложение, которое осуществляет доступ к таблицам БД. Если в приложение местонахождения таблицы указано с помощью псевдонима, то после перемещения БД для обеспечения работоспособности приложения достаточно изменить путь, на который указывает псевдоним. Если же в приложение путь к БД указан в явном виде, то есть без псевдонима, то после перемещения БД нужно перемещать само приложение – вносить изменения в исходный код и заново его транслировать.

Регистрация псевдонима:

1. Воспользуемся приложением SQL Explorer, запускаемым командой **Database►Explore.** В левой части окна приводится список всех зарегистрированных в системе **BDE** баз данных, в правой – свойства текущей базы, выбранной в списке.
2. Создадим псевдоним для базы данных. Для этого выполним команду **Object►New** и в диалоговом окне выбора драйвера укажем значение **Standart**.



3. После щелчка по кнопке ОК в списке появится новый элемент, помеченный зеленым треугольником.



4. По умолчанию формируется имя базы данных **Standard1**, изменим его на **Student**.
5. Убедимся, что в свойствах **Default Driver** (Драйвер по умолчанию) стоит значение Paradox. В свойстве **Path** укажем каталог, в котором хранится наша таблица.
6. Теперь зарегистрированную в системе **BDE** базу сохраним, выбрав для этого **Apply** в контекстном меню объекта Student.
7. На вопрос о необходимости сохранения изменений нажать **Yes**. Теперь таблица доступны из среды **BDE** под именем Student.
8. Закройте **Sql Explorer**.

ЛАБОРАТОРНАЯ РАБОТА 2.

СОЗДАНИЕ ФОРМ В DATABASE DESKTOP В DELPHI.

Цель работы:

1. Изучить начальные этапы создания приложения для работы с базами данных в среде Delphi:

— ознакомиться с компонентами доступа к БД: **TTable, TDataSource;**

— ознакомиться с компонентами управления БД: **TDBGrid, TDBNavigator.**

2. Усвоить ввод и редактирование текста.

Методика создания приложения для работы с базой данных ничем не отличается от методики создания обычной программы: к форме добавляются необходимые компоненты, устанавливаются значения свойств компонентов, разрабатываются необходимые процедуры обработки событий.

Приложение работы с базой данных должно содержать компоненты, обеспечивающие доступ к данным, возможность просмотра и редактирования содержимого полей. Компоненты доступа к данным находятся на вкладке **Data Access** и **BDE** палитры компонентов, а компоненты отображения данных — на вкладке **Data Controls.**

Основные компоненты доступа и управления базами данных:

Компоненты доступа к базам данных:

TTable - обеспечивает взаимодействие с таблицей БД, т.е. компонент **TTable** указывает, откуда брать данные, какие поля будут составлять набор данных. Компонент **TTable** имеет следующие основные свойства:

— **DatabaseName** – база данных;

— **TableName** - имя таблицы;

— **Active** – активация таблицы (значение **True** активирует ее).

Свойство **DatabaseName** определяет базу данных, в которой находится таблица. Это свойство может содержать:

— псевдоним;

— путь для локальных БД;

— путь и имя файла базы данных для **Local InterBase;**

— локальный псевдоним, определенный через компонент **TDatabase**.

Свойство **TableName** определяет имя таблицы базы данных.

TDataSource - определяет связь между базой данных и компонентами управления данными, то есть компонент **TDataSource** является промежуточным звеном между компонентом **Table1**, соединенным с

реальной БД и визуальными компонентами **DBGrid1** и **DBNavigator1**, с помощью которых пользователь взаимодействует с таблицей.

В большинстве случаев, все, что нужно сделать с **DataSource** - это указать в свойстве **DataSet** соответствующий **TTable**. Затем, у визуального компонента вроде **DBGrid** или **DBNavigator** в свойстве **DataSource** указывается **TDataSource**, который используется в настоящее время.

Компоненты управления данными с палитры **Data Contorls**:

TDBGrid  - отображает содержимое таблицы БД в виде сетки, в которой столбцы соответствуют полям, а строки - записям таблицы.

Компонент имеет следующие свойства:

- **Data Source** – содержит ссылку на компонент типа **TDataSource**, служащий источником данных;
- **Editor Mode** – если содержит **true**, пользователь может редактировать ячейку после нажатия клавиши **F2** или **Enter**. Игнорируется, если свойство **Option** включает значение **goEditing** или **goAlwaysShowEditor**;
- **Option** – определяет вид и поведение компонента;
- **dgEditing** – разрешает изменение набора данных;
- **dgAlwaysShowEditor** – автоматически переводит столбец в режим редактирования при его выделение;
- **dgTitles** – показывает заголовки столбцов;
- **dgIndicator** – показывает индикатор текущей строки в самом левом фиксированном столбце;
- **dgColumnResize** – разрешает пользователю вручную изменять ширину столбцов;
- **dgColLines** – показывает разделяющие вертикальные линии;
- **dgRowLines** – показывает разделяющие горизонтальные линии;
- **dgTabs** – разрешает переход от столбца к столбцу с помощью клавиши **Tab**;
- **dgRowSelect** – разрешает выделение цветом всей выбранной строки;
- **dgAlwaysShowSelection** – выделение текущей строки сохраняется, если компонент теряет фокус ввода;
- **dgConfirmDelete** – удаление строки должно подтверждаться;
- **dgCancelOnExit** – если пользователь вставил пустую строку и покинул ее, она не помещается в набор данных;
- **dgMultiSelect** – разрешает множественный выбор строк.

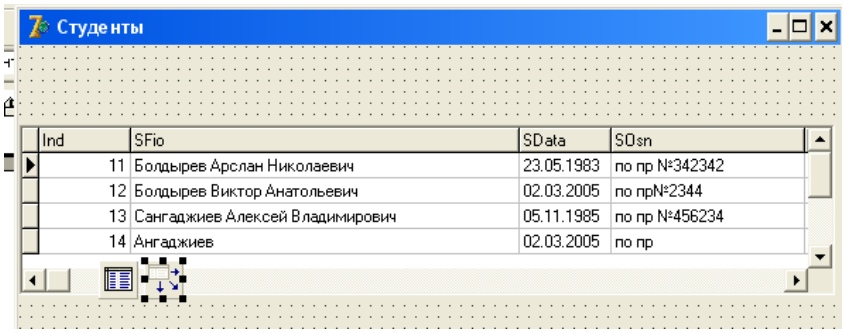
TDBNavigator  - осуществляет перемещение и редактирование записей (вид и назначение кнопок указаны в пункте **DBNavigator**). С помощью свойства **DataSource** компонент связывается с нужным источником данных **TDataSource** – это все, что необходимо для его нормальной работы.

Свойство **ConfirmDelete** управляет отображением диалогового окна с просьбой подтвердить удаление записи (значение **True** этого свойства выводит окно).

Создание приложения:

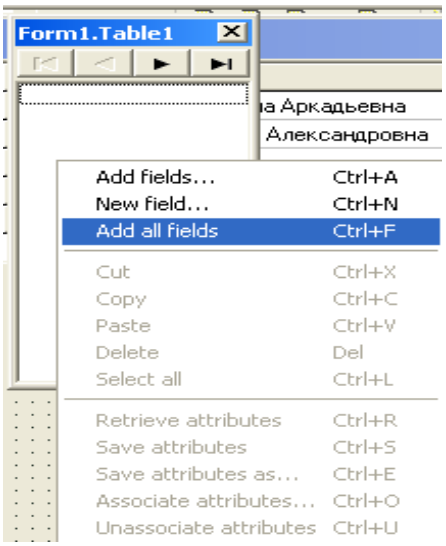
1. Запустить Delphi.
2. В свойстве **Caption** изменить имя формы на *Студенты*.
3. Установить на форму компоненту **TTable**.
4. Определить следующие свойства компоненты **Table1**.
   - — определить псевдоним - выбрать в свойстве **DatabaseName** инспектора объектов псевдоним «Student».
   - — задать имя таблицы - выбрать в свойстве **TableName** таблицу Student.
   - — активизировать таблицу - установить в свойстве **Active** значение **true** (это будет возможно после выполнения пункта 4).
5. Разместить на форму компоненту **DataSource** с закладки **Data Access** и в свойстве **DataSet** инспектора объектов выбрать компоненту **Table1**.
6. Расположить на форме компоненту **DBGrid** с закладки **Data Controls** и в свойстве **DataSource** выбрать **DataSource1**.

Если вы внесли несколько записей, то форма **Студенты** примет вид:



Редактор полей:

Для управления отображением данных таблицы используют специальный редактор полей - **Editor Field**.



Для вызова **Editor Field** следует:

— Дважды щелкнуть по **Table1**.

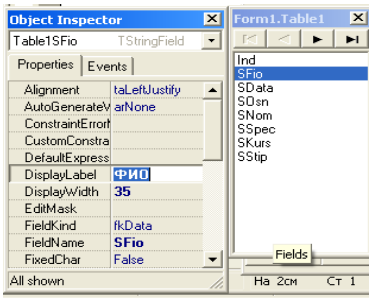
— Для открывшегося окна вызвать контекстное меню и выбрать пункт **Add All Field**, если необходимо добавить все поля таблицы.

— **Add Field** для выбора отдельного поля.

— Редактор полей имеет следующие свойства:

— **DisplayLabel** – задает имя полю*;*

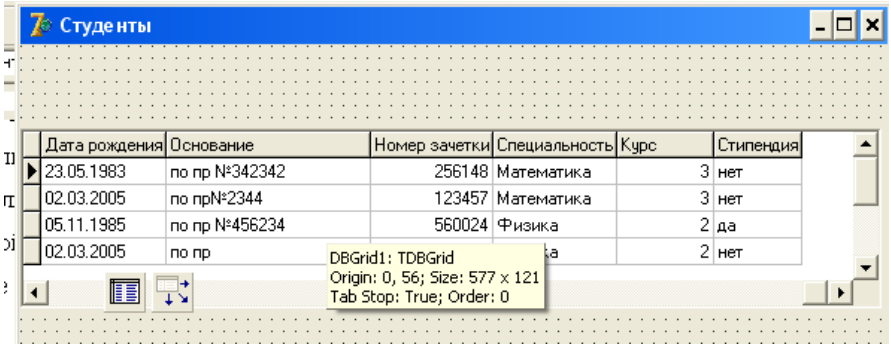
— **DisplayWidth** – определяет количество символов, которое будет выводится в поле;



Определим свойства для полей таблицы *Student.db*.

1. Выбрать в окне редактора полей таблицы поле **SFio** и в свойстве **DisplayLabel** инспектора объектов изменить **SFio** на **ФИО**. Выбрать свойство **DisplayWidth** и заменить размер на 35.
2. Так же поменять свойства других полей таблицы.
3. Для полей логического типа в свойстве **DisplayValues** можно написать варианты для значений **True** и **False**. В поле **SSpec** в этом свойстве написать «**Математика;Физика**» (без пробела, разделяя «;»). Получиться как показано на рисунке.
4. Если возникнет необходимость можно скрыть любое поле, выбрав его и в свойстве **Visible** инспектора объектов установив значение **false.**

После выполненных действий сетка **DBGrid1** будет выглядеть так:



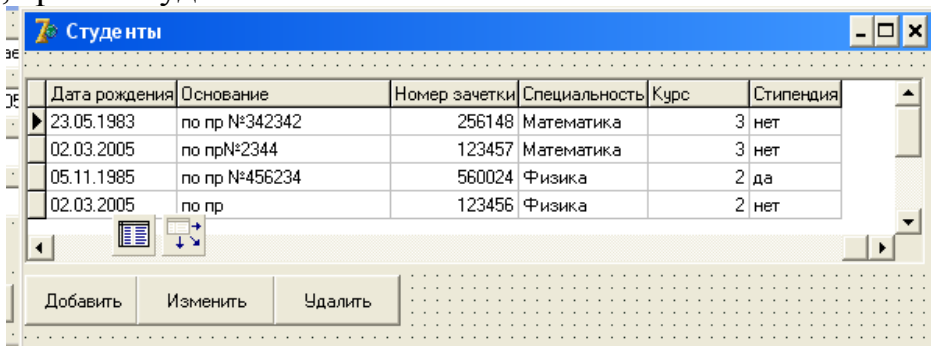
Ввод данных:

Компоненты для организации доступа к таблицам БД позволяют выполнять всевозможные операции с наборами данных: добавлять или удалять записи, перемещаться по ним. При этом следует иметь в виду, что в любой момент времени доступна для выполнения конкретных действий только одна запись, называемая *текущей.* В этой лабораторной работе рассматриваются наиболее часто используемые методы компоненты **Table.**

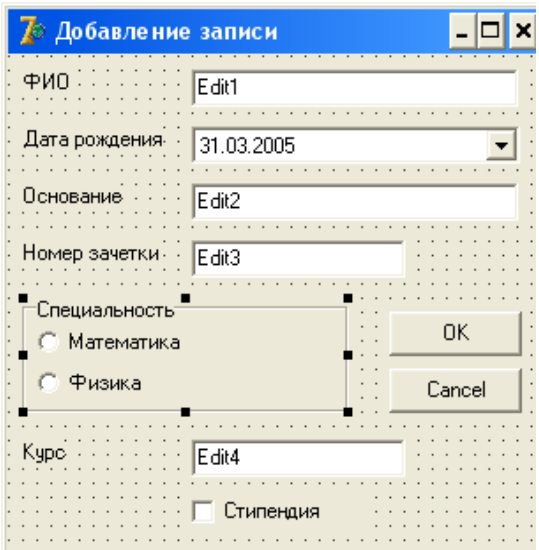
Основные методы для организации доступа компоненты **Table**:

- **Append** – добавить новую запись в конец таблицы.
- **Delete** – удалить текущую строку.
- **Edit** – перейти в режим редактирования. После этого можно изменять значения полей.
- **Insert** – вставить новую строку в таблицу.
- **Post** – принять все изменения.
- **Refresh** – обновить информацию о данных.
- **UpdateRecord** – обновить текущую запись.

1. Откроем созданное приложение.
2. Разместим на форме три компонента **SpeedButton** из палитры **Additional**. Одна из кнопок будет добавлять запись, другая – изменять данные в записи, третья – удалять. Назовем их соответственно.



3. Создадим новую форму, которая будет вызываться нажатием кнопки *«Добавить»*. На форме расположены 4 компонента **Edit**, компонент **DateTimePicker** с закладки **Win32**, компонент **CheckBox** и компонент **RadioGroup.**



4. Текст процедуры для события **OnClick** кнопки *«Добавить»* на форме **Студенты**:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    Form2.ShowModal; //открывает форму «Добавление записи»
```

```
end;
```

5. Текст процедуры для события **OnClick** кнопки *«ОК»* на форме **Добавление записи**:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
Form1.Table1.Insert;
Form1.Table1.FieldByName('SFio').Text:=Edit1.Text;
Form1.Table1.FieldByName('SOsn').Text:=Edit2.Text;
Form1.Table1.FieldByName('SNom').Text:=Edit3.Text;
Form1.Table1.FieldByName('SKurs').Text:=Edit4.Text;
Form1.Table1.FieldByName('SData').AsDateTime:=DateTimePicker1.Date;
if CheckBox1.Checked then
   Form1.Table1.FieldByName('SStip').Text:='да'
     else
       Form1.Table1.FieldByName('SStip').Text:='нет';
          //при нажатии на флажок полю SStip (Стипендия) передается
          //значение True, в противном случае вводится передается
           //значение False
case RadioGroup1.ItemIndex of
0: Form1.Table1.FieldByName('SSpec').Text:='Математика';
1: Form1.Table1.FieldByName('SSpec').Text:='Физика';
end;
if form1.Table1.Modified
   then form1.Table1.Post;
close;
```

Комметарий: в строке *Form1.Table1.Insert* вызывается метод, который допускает вставку новой строки в таблицу, которая находится на форме «**Студенты**». Без вызова этого метода дальнейшая работа по вставке записи в таблицу невозможна. Запись *Form1.Table1.FieldByName('SFio').Text:=Edit1.Text* означает, что текст, который находится в **Edit1** по нажатии кнопки будет перенесен в таблицу на форме «**Студенты**» в новую запись в текстовое поле **ФИО**. Остальные записи в процедуре работают аналогичным образом. Запись *if form1.Table1.Modified then form1.Table1.Post* сохраняет изменения в таблице. *Close* – закрывает форму «**Добавление записи**».

6. По нажатии кнопки **Cancel** осуществляется выход. Тоже и на форме «Редактирование записи».

7. Текст процедуры для события **OnClick** при нажатии клавиши *«Удалить»* на форме **Студенты**:

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
    Table1.Delete          //удаляет текущую запись в таблице
end;
```

Редактирование данных:

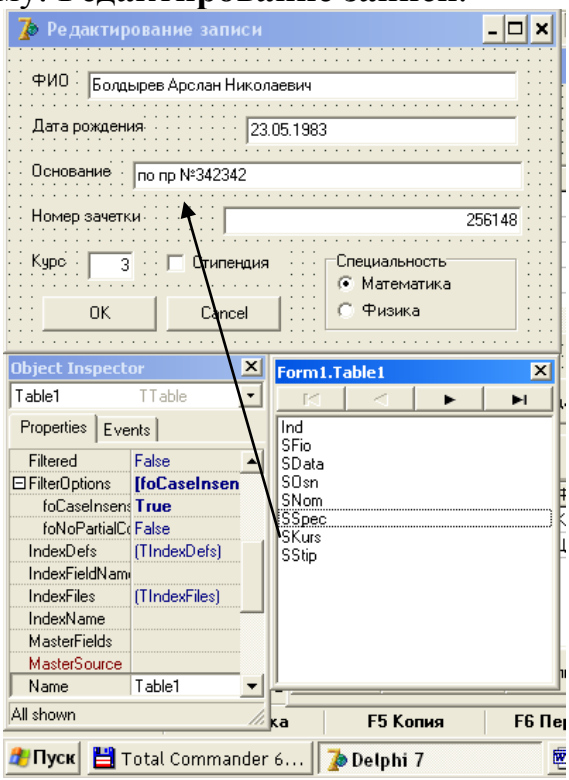
Компоненты, отражающие информацию, делятся на две категории – те, которые не связаны с таблицами БД, и компоненты, связанные с таблицами и обменивающиеся с ними информацией. В первую категорию входят обычные компоненты Delphi. Компоненты второй категории расположены на странице **Data Controls**. Почти каждая из них имеет аналог среди обычных компонент; основные отличия заключаются в том, что они могут работать с данными, хранящимися в БД. К этой группе относится компонента **DBEdit**, которая используется для ввода текстовой однострочной информации.

Чтобы компонент **DBEdit** видел данные из поля таблицы, следует указать в свойствах:

— DataSource – источник данных;
— DataField – поле для редактирования.

Этот компонент с заданными уже свойствами может появиться автоматически при перетаскивании имени поля из окна редактора полей.

1. Создадим новую форму: **Редактирование записи.**



2. В случае перетаскивания поля логического типа, на форме автоматически устанавливается компонента **DBCheckBox**, что не всегда удобно.
   2.1. Установим на форму компоненту **DBRadioGroup** с закладки **Data Control**.
   2.2. В свойстве **DataSource** укажем **DataSource1**.
   2.3. В свойстве **DataField** укажем **SSpec**.
3. Текст процедуры для события **OnClick** при нажатии клавиши *«Изменить»* на форму **Студенты**:

```
begin
        Form3.ShowModal //вызов Form3
end;
```

4. Текст процедуры для события **OnClick** при нажатии клавиши *«Сохранить»* на форму **Редактирование**:

```
begin
        if form1.Table1.Modified
                then form1.Table1.Post;        //все изменения в таблице сохраняются
        close;
end;
```

5. Пользователь имеет возможность редактировать записи в таблице напрямую. Чтобы это предотвратить используется свойство компоненты **DBGrid** ***dgEditing***. Нужно выделить **DBGrid1** и в свойстве ***Options►dgEditing*** инспектора объектов поставить **false**.

ЛАБОРАТОРНАЯ РАБОТА 3-4.

НАЧАЛЬНЫЕ ЭТАПЫ СОЗДАНИЯ ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ В СРЕДЕ DELPHI. ПОИСК ПОЛЕЙ С ПОМОЩЬЮ МЕТОДА LOOKUP.

Цели работы:

1. Ознакомиться с сортировкой записей в базе данных.
2. Поиск полей с помощью методов **Locate** и **Lookup**;

Сортировка:

Порядок расположения записей в таблице БД может быть неопределенным. По умолчанию записи не отсортированы или сортируются, например, для таблиц **Paradox** по ключевым полям, а для таблиц **dBase** в порядке их поступления в файл таблицы.

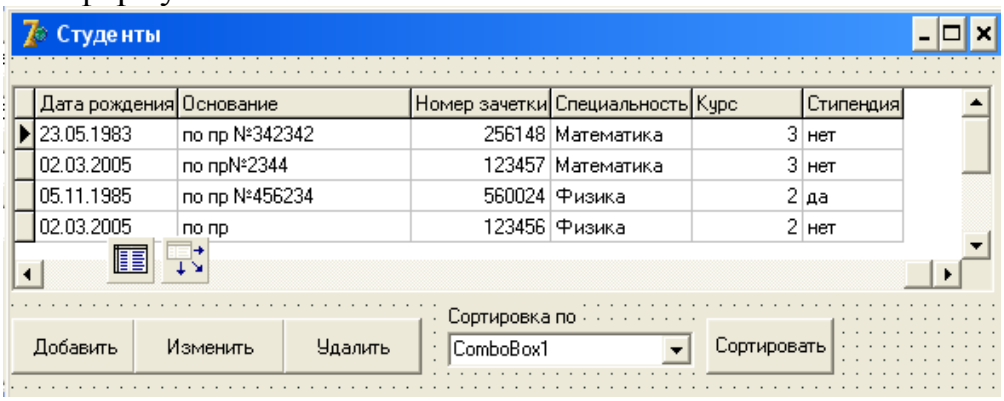
С отсортированными записями набора данных работать более удобно. Сортировка заключается в упорядочивании записей по определенному полю в порядке возрастания или убывания содержащихся в нем записей.

Сортировка набора данных **TTable** выполняется автоматически по текущему индексу. При смене индекса происходит переупорядочивание записей. Таким образом, возможна по полям, для которых создан индекс. Для сортировки по нескольким полям нужно создать индекс, включающий эти поля.

Задать индекс, по которому выполняется сортировка записей, можно с помощью свойств:

- **IndexName** – указывается имя индекса, установленное при его создании;
- **IndexFieldName** – указываются имена полей, образующий соответствующий индекс.

1. Откроем приложение.
2. Добавим на форму компоненты **ComboBox** и **Button**.



3. В свойстве **Items** компоненты **ComboBox** запишем параметры сортировки: Фамилия, Специальность, Курс, Дата рождения, Номер зачетки.

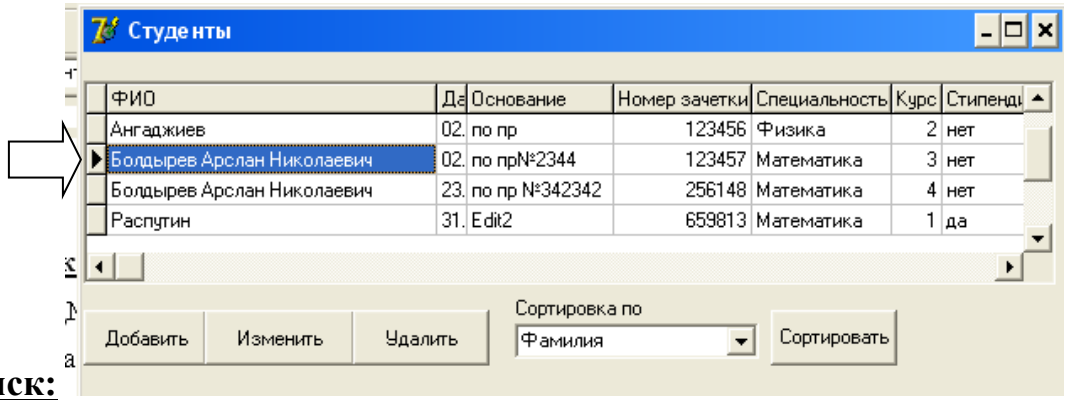
a. Условия сортировки задаются вторичными индексами. То есть сортировка по фамилии происходит по вторичному ключу **IDFio** так как в него первым входит поле **SFio**. Для того, чтобы сортировка проходила по выбранным параметрам необходимо вхождение соответствующих полей в разные вторичные ключи.

4. Текст процедуры для события **OnClick** при нажатии кнопки *«Сортировка»* на форме **Студенты**:

```
begin
  Case ComboBox1.ItemIndex of
    0: Table1.IndexFieldNames:='SFio'; //при выборе строки «Фамилия»
                  //сортировка идет по вторичному индексу IDFio
    1: Table1.IndexFieldNames:='SSpec';
    2: Table1.IndexFieldNames:='SKurs';
    3: Table1.IndexFieldNames:='SData';
    4: Table1.IndexFieldNames:='SNom';
  end;
end;
```

Замечание: во вторичный индекс *IDFio* входят поля: **SFio**, **SKurs, SSpec**. То есть при совпадении фамилии сортировка идет уже по курсу и т.д.

Пример:



Поиск:

Метод **Locate** ищет первую запись, удовлетворяющую критерию поиска, и если такая запись найдена, делает ее текущей. В этом случае в качестве результата возвращается значение **True**. Если запись не найдена, возвращается значение **False** и курсор не меняет своего положения.

function Locate (**const** KeyFields: **String**; **const** KeyValues: Variant;
Options: TLocateOptions): Boolean;

Список полей, по которым ведется поиск, задается в параметре **KeyFields**, поля разделяются точкой с запятой. Параметр **KeyValues** типа

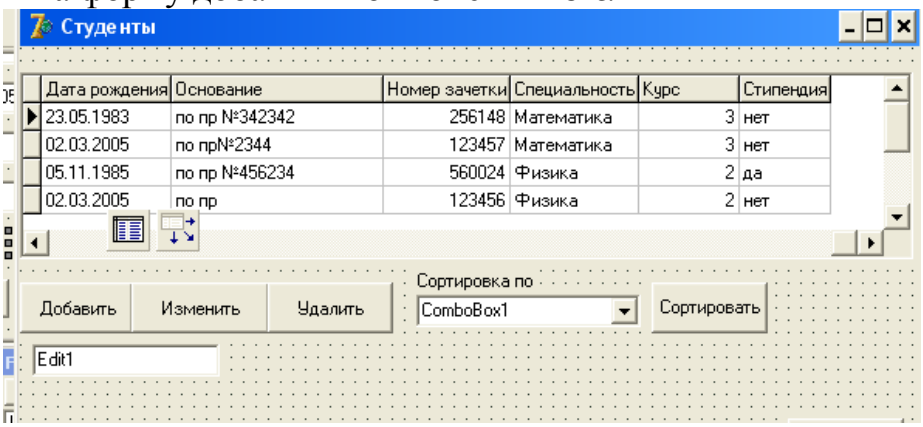
Variant указывает значение полей для поиска. Если поиск ведется по одному полю, то параметр содержит одно значение, соответствующие типу поля, заданного для поиска.

Параметр **Options** позволяет задать значение, которое обычно используется при поиске строк. Этот параметр принадлежит к множественному типу **TLocateOptions** и принимает комбинации следующих значений:

— *LoCaseInsensitive* –регистр букв не учитывается;

— *LoPartialKey* – допускается частичное совпадение.

1. На форму добавить компонент **Edit**.



2. Текст процедуры для события **OnChange** компонента **Edit** на форме **Студенты**:

```
begin
  table1.Locate('SFio',Edit1.Text,[loPartialKey]);
end;
```

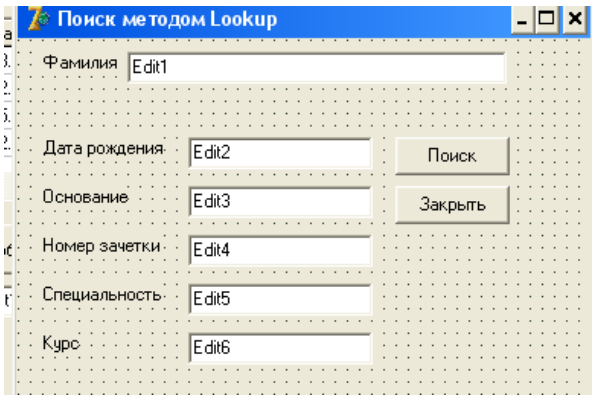
3. Поиск записи по фамилии организован. Регистр букв не учитывается.

Метод **Lookup** находит запись, удовлетворяющую условию поиска, но не делает ее текущей, а возвращает значения некоторых ее полей. Независимо от результата поиска записи указатель текущей записи в НД не изменяется. В отличие от метода **Locate**, метод **Lookup** осуществляет поиск только на точное соответствие критерию поиска значения поля поиска записи.

function Lookup (**const** KeyFields: **String; const** KeyValues: Variant;
const ResultFields: **String**): Variant;

В параметре **ResultFields** перечисляются поля, значения которых требуется получить в случае успешного поиска. Тип результата – **Variant** или вариантный массив.

1. Добавить на главную форму новую кнопку «Поиск».
2. Открыть новую форму и ввести компоненты как показано на рисунке. Эта форма вызывается нажатием кнопки поиска на главной форме.



3. Поиск будет осуществляться по фамилии, введенной в компоненте Edit1 после нажатия кнопки на форме «Поиск методом Lookup».

```
procedure TForm5.Button1Click(Sender: TObject);
var  LookupResult: Variant;
begin
LookupResult:=Form1.Table1.Lookup('SFio', Edit1.Text, 'SData; SOsn; SNom;
SSpec; SKurs');      //ищем поля  'Дата рождения'
//'Основание', 'Номер зачетки', 'Специальность', 'Курс'
            if VarIsArray (LookupResult) then
                begin
                Edit2.Text:=LookupResult[0]; //записывает значения
                Edit3.Text:=LookupResult[1]; // в искомых полях в
                Edit4.Text:=LookupResult[2]; //соответствующие
                Edit5.Text:=LookupResult[3]; //компоненты
                if Edit5.Text='False' then
                    Edit5.Text:='Физика'   //поиск полей логического типа
                        else Edit5.Text:='Математика';
                Edit6.Text:=LookupResult[4];
                end;
        end;
```

ЛАБОРАТОРНАЯ РАБОТА 5.

ФИЛЬТРАЦИЕЙ ЗАПИСЕЙ В DELPHI.

Цели работы:

1. Ознакомиться с фильтрацией записей.

Фильтрацию можно отнести к одному из методов поиска. Потому что фильтрация – выбор из набора данных только тех записей, которые удовлетворяют конкретным условиям. Например, можно указать отображение только записей, в которых поле «*Фамилия*» содержит значение «*Иванов*». Применение фильтра к набору данных определяется свойством **Filtered** логического типа. Значение **True** определяет применение в качестве фильтра выражения, указанного в свойстве **Filter**:

Поле [Оператор сравнения] 'Значение'

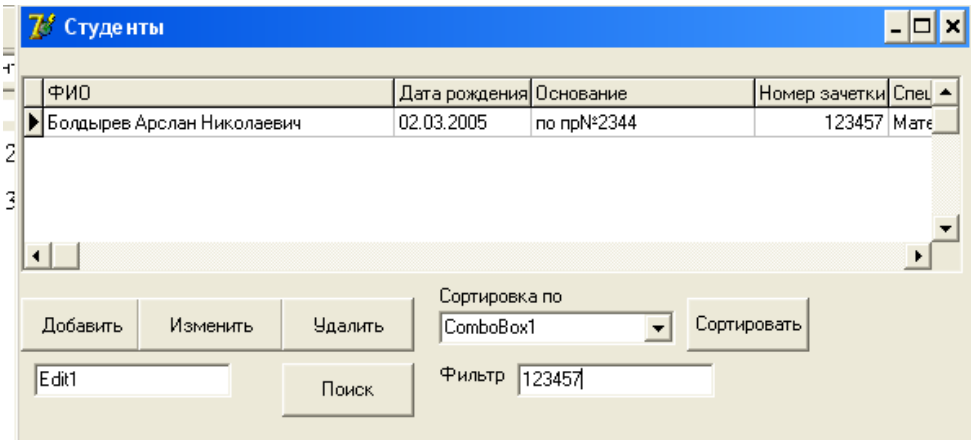
Например, если отобразить все записи, в которых поле «*Фамилия*» равно значению «Сидоров», то нужно указать:

Table1.Filter:='Фамилия=''Сидоров''';

Фильтрация записей:

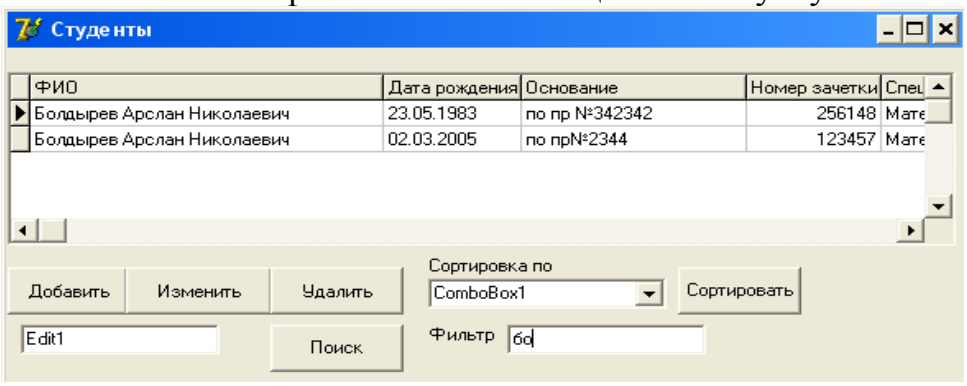
1. Открыть приложение.
2. Добавить на форму компоненту **TEdit**.
3. Текст процедуры для события **OnChange**:

```
begin
  Table1.Filtered:=true;          //включение фильтрации
  Table1.Filter:='SNom = '+Edit2.Text;
        //задает критерий фильтрации
end;
```



4. Этот способ фильтрации пригоден только для числовых полей.
5. При применении фильтра можно указать свойства:
   a. **foCaseInsensitive** – нечувствительность к регистру букв;
   b. **foNoPartialCompare** – поиск на точное соответствие.

6. Для фильтрации текстовых полей, например по полю «Фамилия» необходимо изменить текст процедуры.
   procedure TForm1.Edit2Change(Sender: TObject);
   begin
   Table1.Filtered:=true;
   Table1.Filter:='SFio='+#39+Edit2.Text+''+#39;*
   end;
7. В этом случае фильтрация проходит по текстовому полю. Знак «#39» означает знак апострофа, так как ввод фамилии при использовании фильтра происходит в апострофах. А символ «*» означает любые символы, то есть при вводе только одной буквы на экране появятся все фамилии начинающиеся на букву.



ЛАБОРАТОРНАЯ РАБОТА 6

СВЯЗИ МЕЖДУ ТАБЛИЦАМИ В DELPHI

Цели:

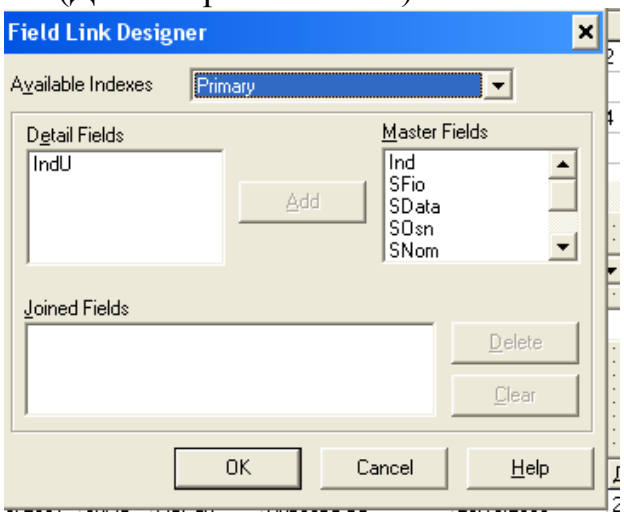
1. Усвоить возможность связывания таблиц.

Установка связи между таблицами:

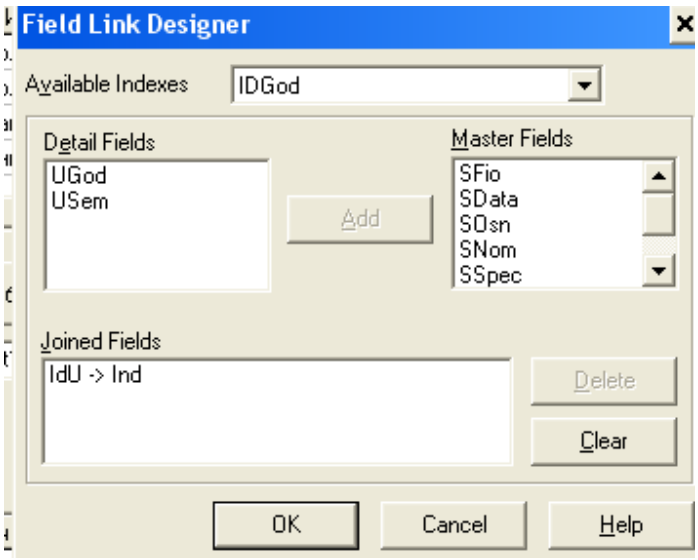
Для демонстрации связи между таблицами необходимо создать еще одну таблицу.

Создайте таблицу успеваемости студентов. В нее войдут поля: учебный год, сессия (зима или лето), предмет, ФИО преподавателя, дата аттестации по предмету, дата сдачи, оценка.

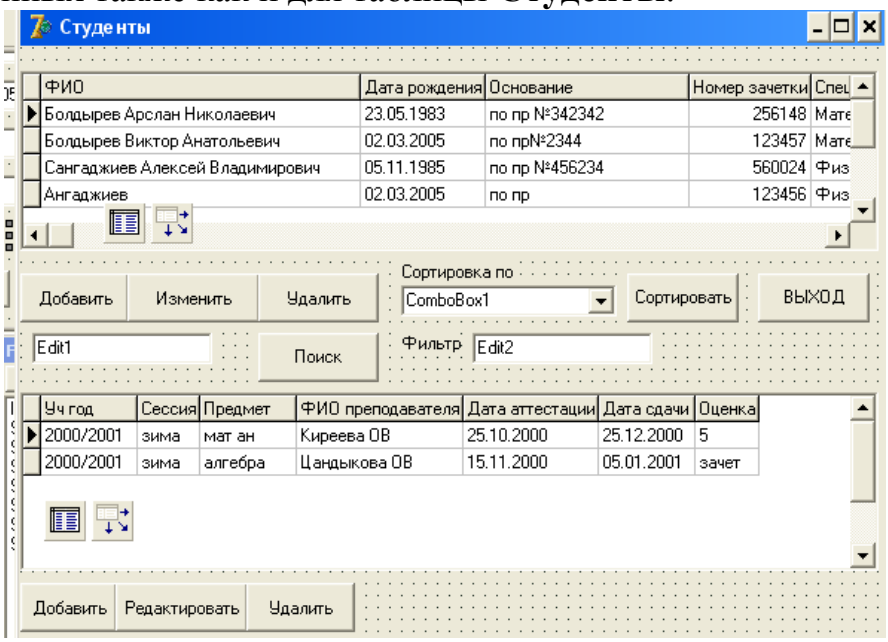
1. Эта таблица будет дочерней для таблицы **Студенты**. В таблице **Успеваемость** надо ввести дополнительно числовое поле и определить его вторичным ключом.
2. Разместите таблицу успеваемости на форме **Студенты**.
3. В свойстве компоненты **Table2 Master Source** написать **Data Source1.** Это означает, что вторая таблица станет дочерней для первой.
4. Двойным щелчком по свойству **Master Fields** вызовем окно Field Link Designer (Дизайнер поля связи):



4. Выбрать в списке **Available Index (Доступные индексы)** из окна **Field Link Designer** индекс **IDGod** (это вторичный индекс второй таблицы).
5. В левом списке **Detail Field** выделить **IdU**, а в правом списке **Master Field *(Основа)*** выделить **Ind**.
6. Нажать на кнопку **Add (Добавить)** и закрыть окно.



7. Таким образом, между таблицами установилась связь. Она называется *связь один ко многим*.
8. К дочерней таблице добавить кнопки для ввода и редактирования данных также как и для таблицы **Студенты**.



ЛАБОРАТОРНАЯ РАБОТА 7

СОЗДАНИЕ ОТЧЕТОВ В DELPHI

Цели работы:

1. Ознакомиться с возможностью создания отчета.

Создание отчетов:

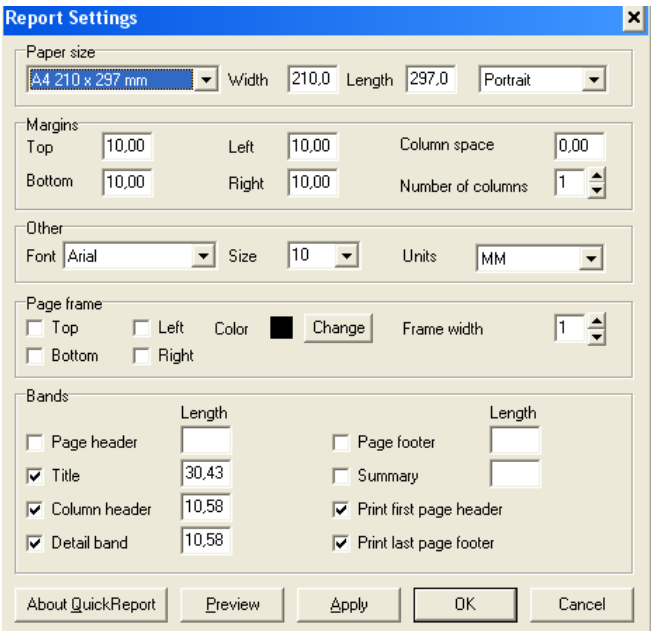
Отчет — это печатный документ, содержащий записи БД. В Delphi для создания отчетов служит генератор отчетов **QuickReport**, содержащий обширный набор компонентов. Компоненты, предназначенные для создания отчетов, находятся на закладке **QReport** палитры компонентов.

Главным элементом отчета является компонент-отчет **QuickRep**, представляющий собой основу, на которой размещаются другие компоненты. Компонент **QuickRep** обычно размещается на отдельной форме, предназначенной для создания отчета.

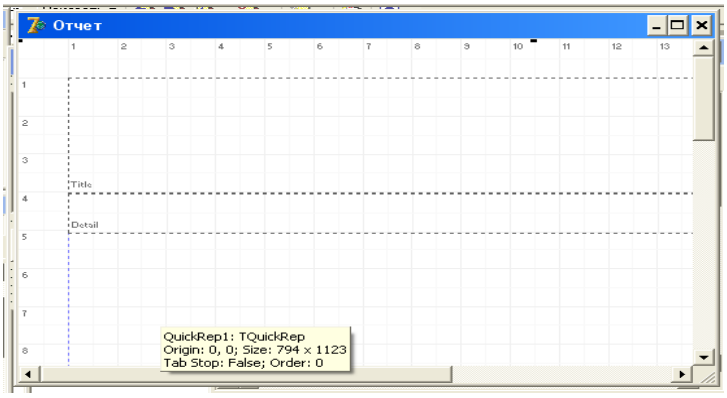
Свойства компоненты **QuickRep**:

- **Bands** – здесь указываются компоненты размещаемые в **QuickRep**.
- **DataSet** – здесь указывается набор данных, из которой отчет будет брать данные.
- **Frame** – здесь указывается параметры рамки.
- **Options** – здесь доступны три параметра. Если **FirstPageHeader** равно **true**, то заголовок печатается только на первой странице отчета. Если **LastPageFooter** равен **true**, то нижний колонтитул печатается только на последней странице отчета. Если установить свойство **Compression** в **true**, то отчет будет сохраняться в сжатом виде.
- **ReportTitle** – здесь находится заголовок печатаемого документа.
- **SnapToGrid** – нужно ли выравнивать компоненты по установленной сетке.
- **Zoom** – масштаб отображения данных.

Настройку параметров отчета можно выполнить с помощью окна **Report Settings**, вызываемый двойным щелчком мыши по компоненте **QuckRep**. Предпочтительно пользоваться именно этим окном, так как здесь всегда можно просмотреть будущий результат.



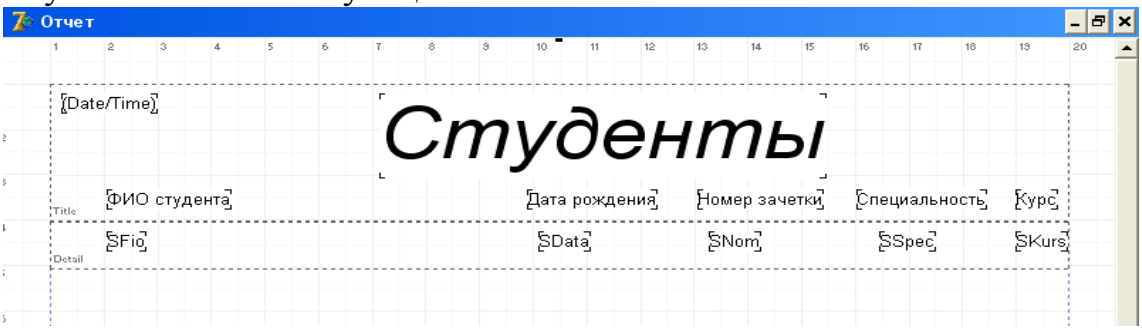
1. Открыть таблицу «**Студенты**».
2. Добавить на главную форму кнопку «**Создание отчета**».
3. Создать новую форму «**Отчет**», которая будет вызываться нажатием на кнопку «**Создание отчета**».
4. На форму установить компоненту **QuickRep** с закладки **QReport**. Выделить этот компонент и в объектном инспекторе включить параметры **HasTitle** и **HasDetail** свойства **Bands**.



5. Расположим компоненты в секциях **QuickRep1**, которые будут отображать нужную информацию отчета. На закладке **QReport** палитры компонентов доступны следующие компоненты, которые можно расположить в этих разделах:
   - **QRLabel** – надпись. Этот компонент похож на стандартный компонент **TLabel** и просто отображает нужные данные.
   - **QRDBText** – данные. Этот компонент тоже похож на **TLabel**, только он предназначен для отображения значения какого-либо поля из базы данных.
   - **QRSysData** – системная информация. Это опять копия **TLabel** только с возможностью отображать системную информацию – дату, время, номер страницы, номер строки в таблицы, общее количество страниц и т.д.

— **QRImage** – картинка. Компонент схожий с **TImage**.

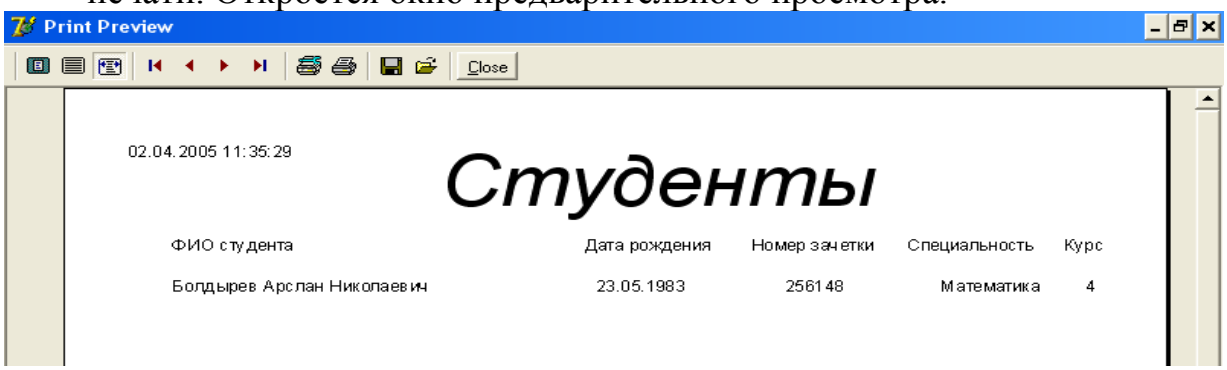
6. Увеличить область заголовка **Title**. В верхний угол поместите один компонент **QRSysData**. Выделить его и в свойстве **Data** выбрать значение **qrsDateTime**. Теперь этот компонент будет отображать в правом, верхнем углу дату распечатки документа.
7. В центре области **Tittle** установить компонент **QRLabel**, увеличь шрифт в свойстве **Font** и написать в свойстве **Caption** текст «**Студенты**».
8. Расположить в области **Tittle** компоненты **QRLabel** и дать им заголовки: ФИО, Дата рождения, Номер зачетки, Специальность, Курс.
9. Перейти к области **Detail**. Под заголовками поставить пять компонентов **QRDBText**. Установить в свойстве **DataSet** компонентов **QRDBText** набор данных - **Form1.Table1**, а в свойстве **DataField** для **QRDBText1** указать **SFio**. У всех остальных компонентов **QRDBText** указать соответствующие имена полей.



10. Перейти в главный модуль и по нажатию кнопки “Печать” написать следующий код.

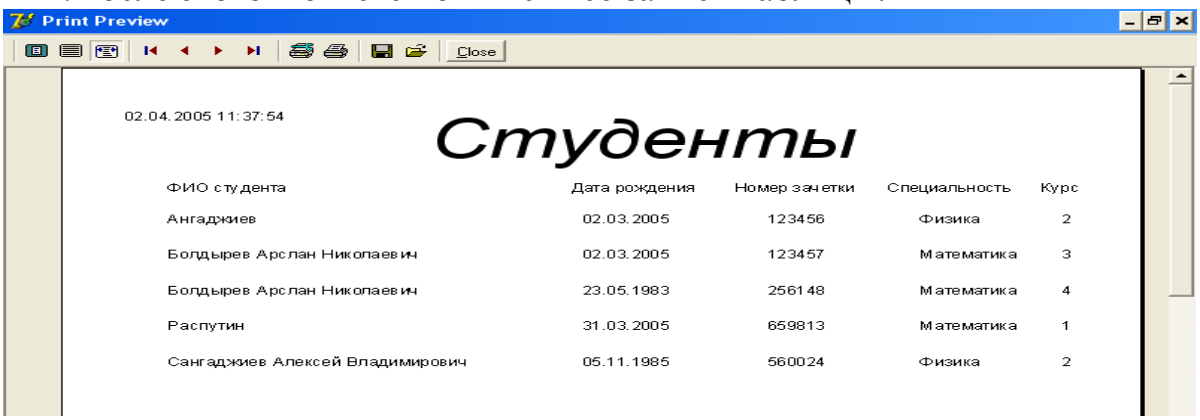
```
procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
    Form4.QuickRep1.Preview; //вызывается метод Preview
        //компонента QuickRep. Этот метод показывает окно
        //предварительного просмотра созданного  документа.
end;
```

11. Запустить программу, выделить какую-нибудь строку и нажать кнопку печати. Откроется окно предварительного просмотра.

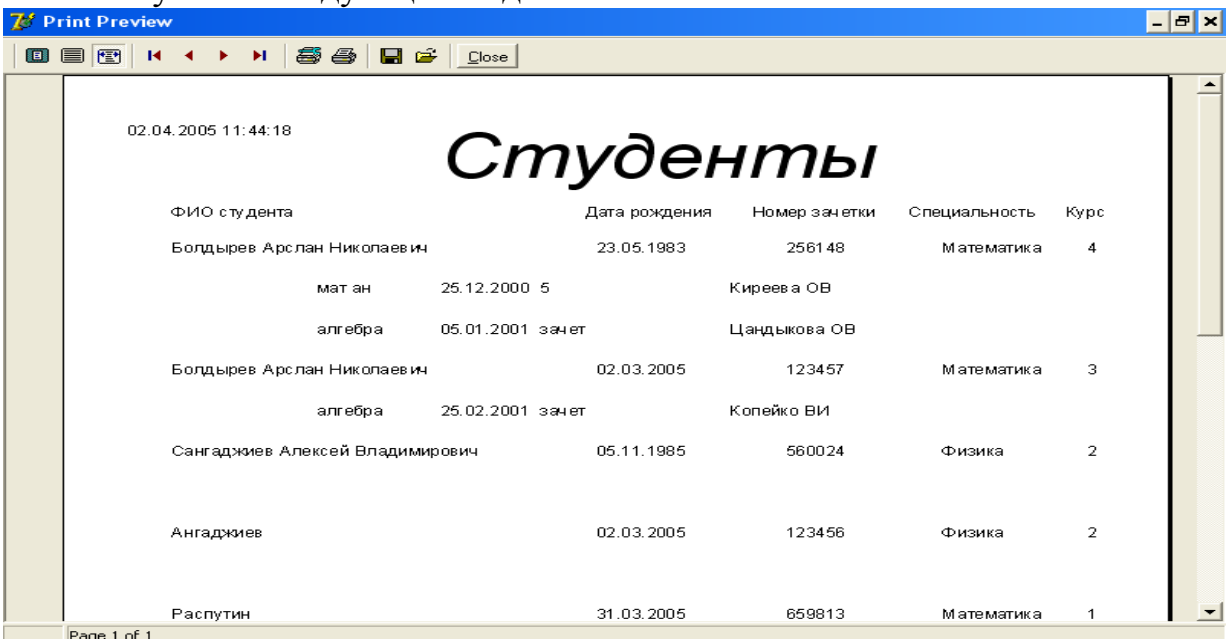


12. Выделить компонент **QuickRep1** и в свойстве **DataSet** указать таблицу **Form1.Table1**.

13. Если сделать это, то компонент **QuickRep1** автоматически будет перебирать все записи из этой таблицы и использовать их в компонентах, которые стоят в блоке **DetailBand1**.
14. После этого в отчете появятся все записи таблицы:



15. Установить на форму отчета компонент – **QRSubDetail** с закладки **QReport**. Этот компонент предназначен для перебора данных относящихся к подчиненным таблицам.
16. Установить следующие свойства: **DataSet** – **Form1.Table2**, чтобы связать блок с таблицей **Uspevaemost.db**, которая является подчиненной к основной **Studenti.db**.
17. В свойстве **Master** нужно указать главный компонент с основными данными. Выбрать в этом свойстве **QuickRep1**.
18. Расположить на компоненте **QRSubDetail** компоненты **QRDBText** в свойстве указав, к каким полям подчиненной таблицы они обращаются.
19. Получится следующий вид отчета:



ЛАБОРАТОРНАЯ РАБОТА 8.

СОЗДАНИЕ SQL-ЗАПРОСОВ В DELPHI

Цель работы: Научиться создавать SQL-запросы к базам данных в среде программирования Delphi, используя компонент ADOQuery.

Постановка задачи. Окончательный вид окна просмотра SQL-запроса, приведен на рисунке 1. Окно содержит три именованные рабочие области.

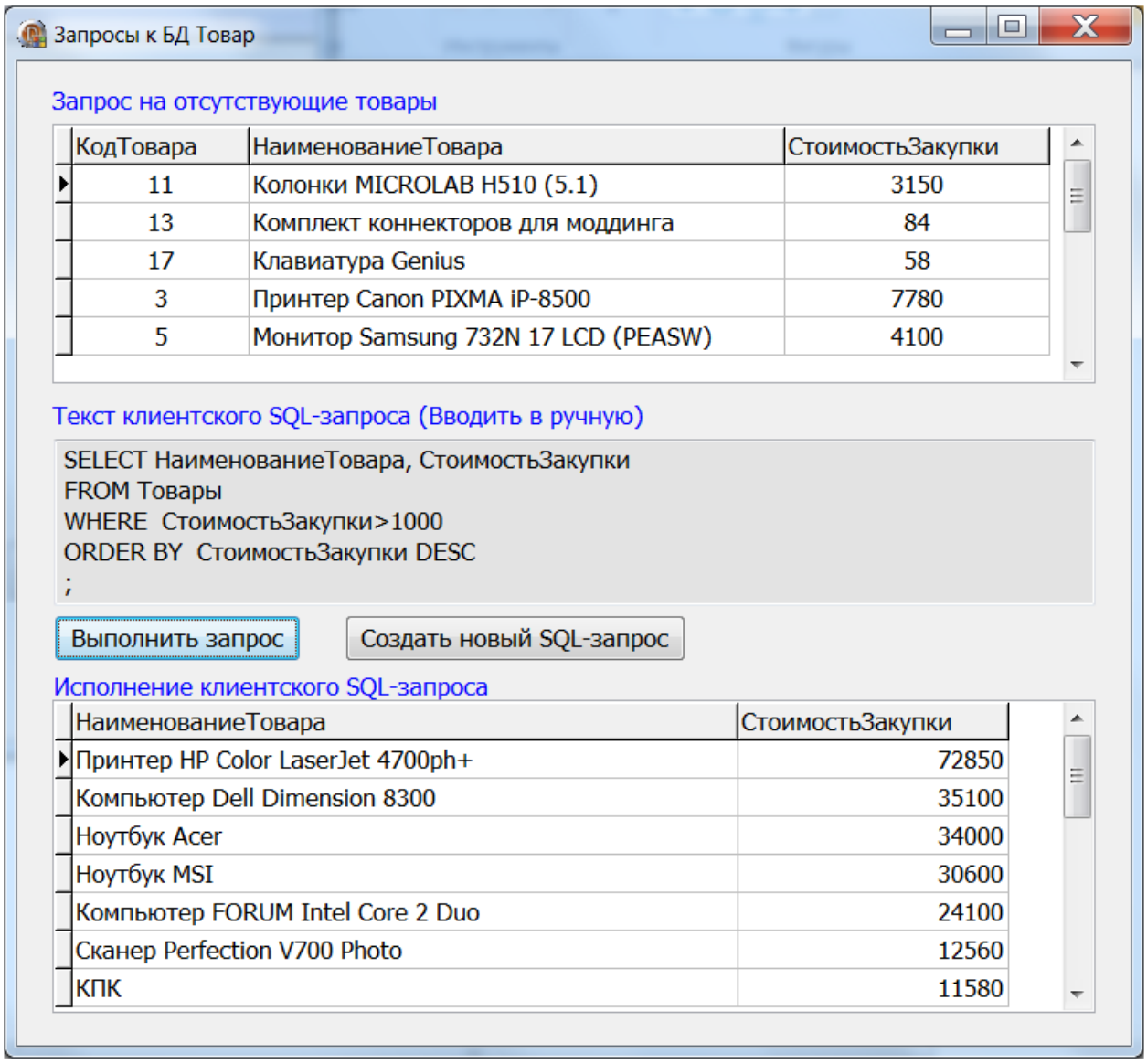


Рисунок 1 – Вид окна «Запросы к БД Товар»

В первой области (сетка DBGrid1) выводится готовый SQL-запрос на отсутствующие товары. Код этого запроса прописан в программе.

Во вторую область (компонент Memo) в ручном режиме вводится клиентский SQL-запрос. Нажимается кнопка «Выполнить запрос», после чего

в третьей области (сетка DBGrid2) появляются записи удовлетворяющие условию введенного клиентского запроса.

1 Создание нового окна «Запросы к БД»

1.1 Создаем **Form5** (File -> New-> Form-Delphi), сохраняем как **Unit5**.

1.2 Устанавливаем свойство **Form5.Caption = Запросы к БД**.

1.3 Знакомим **Form5** с **Form1** и **DatMod** (с помощью **File->Use Unit..**).

1.4 Переходим на **Form1**. В меню приложения добавляем новую закладку **Запросы.** Для этого дважды кликаем на компонент **MainMenu1** и в окне создания меню выбираем нужные места (выставляя курсор) и в соответствующие свойстве **Caption** пишем заголовки пунктов (см. рисунок 2).

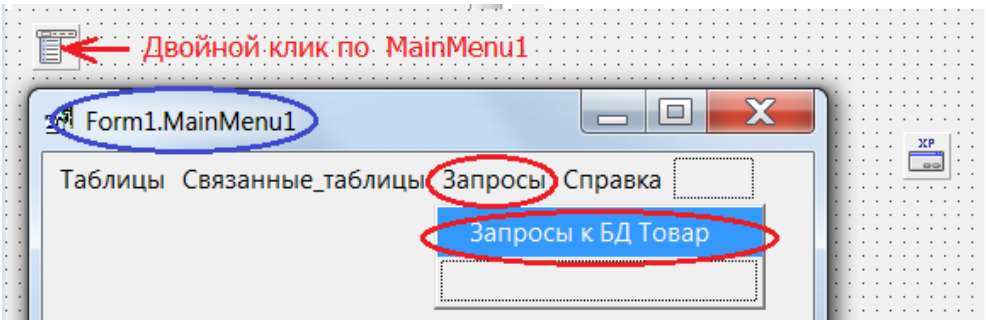


Рисунок 2 – Настройка компонента MainMenu1

1.5 Дважды кликаем по пункту меню **Запросы к БД Товар** из вкладки **Запросы** (см. рисунок 2)**.** В обработчике событий пишем код:

```
procedure TForm1.N7Click(Sender: TObject);
begin
  Form5.Show;     // Метод Show выводит на экран окно Form5
end;
```

1.6 Переходим на **Form5**. Положите на **Form5** компонент **Memo** (закладка Standard), две кнопки **Button** (закладка Standard) с надписями **Выполнить запрос** и **Создать новый SQL-запрос**, две сетки **DbGrid** (закладка Data Controls) и три метки **Label** (закладка Standard) с соответствующими надписями синего цвета. Расположите компоненты как на рисунке 1.

1.7 Сохраняем проект (**File -> Save All)** и компилируем (**F9)**.

2 Создание запроса «Отсутствующие товары»

2.1 На модуль **dm** положите компонент **ADOQuery** ((закладка dbGo) и источник **DataSource** (закладка DataAccess), которому дайте имя **dsQuery1**.

2.2 Свяжите **ADOQuery1** с **ADOConnection1,** как показано на рисунке 3.

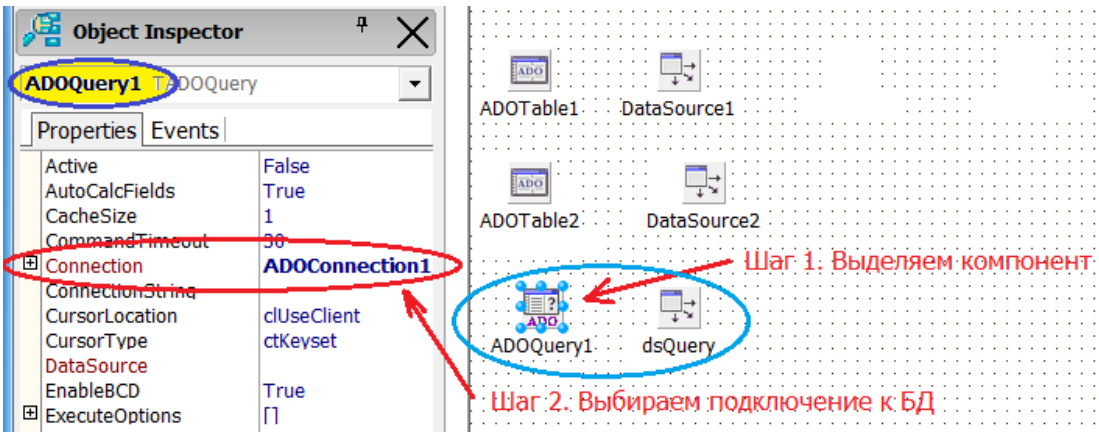
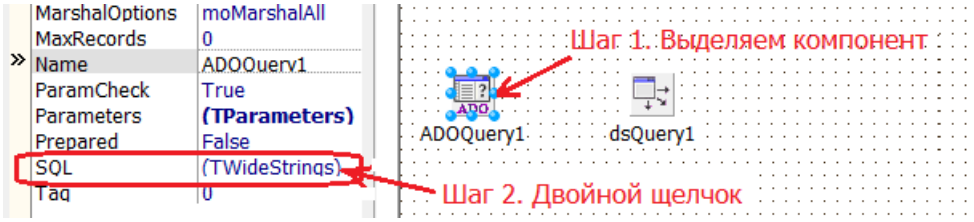


Рисунок 3 – Установка свойств **ADOQuery1**

2.3 В инспекторе **dsQuery1** установите свойство **DataSet** = **ADOQuery1**.

2.4 Переходим на **Form5** и выделяем **DbGrid1**. В инспекторе **DbGrid1** установите свойство **DataSource = dsQuery1**.

2.5 Переходим на **dm**. Выделяем компонент **ADOQuery1** и кликаем в свойстве **SQL** по строке **TWideStrings**.



Появляется окно редактора SQL-запросов. Записываем туда код для отсутствующего товара (см. рисунок 4). Нажимаем OK.

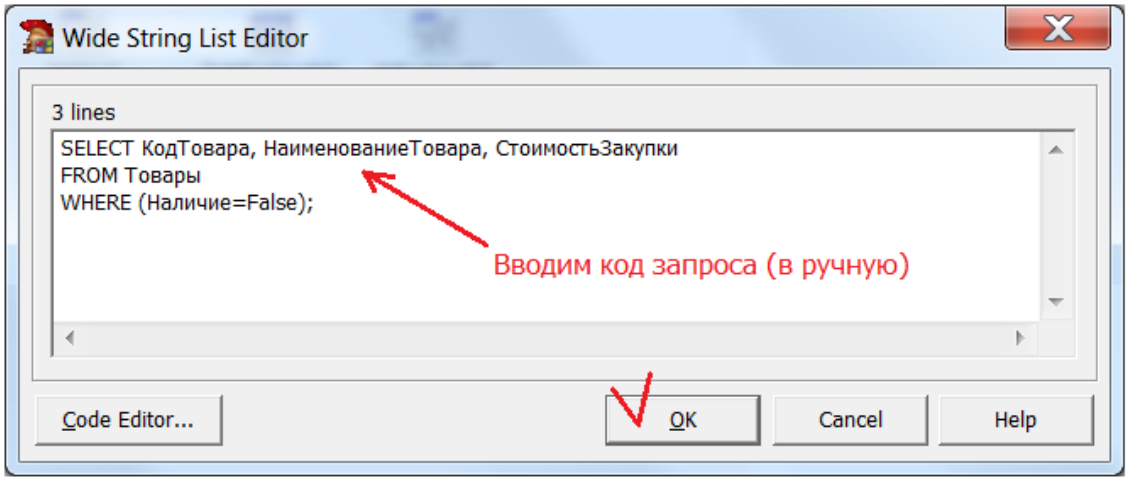


Рисунок 4 – Окно редактора SQL-запросов с запросом

2.6 В Инспекторе **ADOQuery1** устанавливаем свойство: **Active = True**. Это свойство устанавливается в последнюю очередь, в противном случае,

возможно возникновение ошибки «ADOQuery1: Missing SQL property», если свойство ADOQuery1.SQL пусто (рисунок 5).

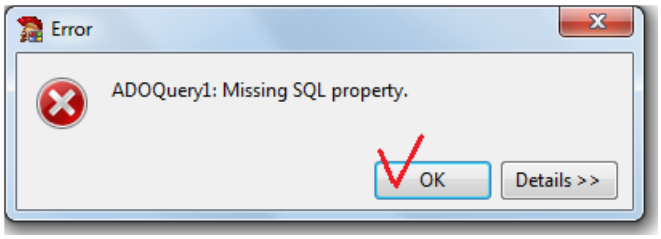


Рисунок 5 – Ошибка отсутствия запроса в свойстве

2.7 Сохраняем проект (**File -> Save All)** и компилируем (**F9)**.

3 Создание клиентского запроса

3.1 На модуль **dm** положите компонент **ADOQuery** ((закладка dbGo) и источник **DataSource** (закладка DataAccess), которому дайте имя **dsQuery2**.

3.2 Свяжите **ADOQuery2** с **ADOConnection1,** как показано на рисунке 6.

3.3 В Инспекторе **dsQuery2** установите свойство **DataSet** = **ADOQuery2**.

3.4 Переходим на **Form5** и выделяем **DbGrid2**. В инспекторе **DbGrid2** установите свойство **DataSource = dsQuery2**.

3.5 По щелчку на **Button1 (Выполнить запрос)** SQL-запрос из **Memo1** следует передать в свойство **ADOQuery2.SQL** где он автоматически выполняется. Создаем обработчик этого события (**OnClick**):

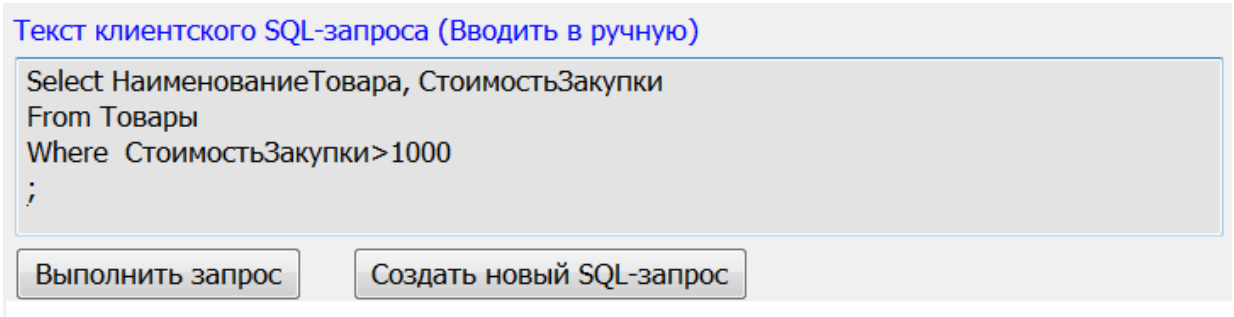
```
// Кнопка "Выполнить запрос"
procedure TForm5.Button1Click(Sender: TObject);
begin
 try  // Если при обработке не произойдет ошибки
   // закрыть ADOQuery2 перед его изменением
   dm.ADOQuery2.Active:= false;
   // Очистка полей свойства SQL компонента ADOQuery2
   dm.ADOQuery2.SQL.Clear;
   //Передать запрос из Memo1 в свойство ADOQuery2.SQL
   dm.ADOQuery2.SQL.Text:=Memo1.Lines.Text;
   // Активизировать ADOQuery2. SQL-запрос выполняется.
   dm.ADOQuery2.Active:= true;
 except // в случае ошибки (например, запрос не верен)
   on e:Exception do  // Универсальная обработка ошибок
    ShowMessage('Не удалось выполнить запрос.');
 end;
end;
```

Пояснение. В Memo1 вводится SQL запрос, после чего нажимается кнопка «**Выполнить запрос».** В случае какой-либо ошибки, например, выражение не соответствует правилам SQL-запроса, на экран выводится типичное сообщение типа «OK» с надписью «Не удалось выполнить запрос».

3.6 Сохраняем проект (**File -> Save All)** и компилируем (**F9)**.

4 Наводим «красоту» на форме Form5

4.1 Переходим на **Form5.** Сделаем так, чтобы при первом открытии этой формы, в **Memo1** был заложен следующий клиентский запрос:



В Инспекторе компонента **Form5** вызываем событие **OnCreate** (или двойной щелчок по **Form5**). Отметим, что событие **OnCreate** генерируется только один раз при создании формы.

В открывшемся окне **Code** вводим код:

```
// Событие OnCreate генерируется только при создании формы
procedure TForm5.FormCreate(Sender: TObject);
begin
 Memo1.Lines.Clear; // Ощищаем строки Memo
 // Заполняем строк Memo1 начиная с первой
 Memo1.Lines.Add('Select НаименованиеТовара, СтоимостьЗакупки');
 Memo1.Lines.Add('From Товары');   //Во вторую строку пишем From
 Memo1.Lines.Add('Where  СтоимостьЗакупки>1000'); //Третья строка
 Memo1.Lines.Add(';'); // В третья строку ";"
end;
```

4.2 Сохраняем проект (**File -> Save All)** и компилируем (**F9)**.

4.3 Переходим на **Form5.** В Инспекторе **Button2** устанавливаем свойство: **Caption = Создать новый SQL-запрос**.

4.4 Создайте обработчик события **OnClick** кнопки **Button2:**

```
// Кнопка Создание нового SQL-запроса
procedure TForm5.Button2Click(Sender: TObject);
var
  p:TPoint; //Переменная для координат x,y курсора в Memo
begin
 // Создаем заготовку для нового SQL-запроса
 Memo1.Lines.Clear; // Ощищаем строки Memo
 Memo1.Lines.Add('Select    '); // В первую строку пишем Select
 Memo1.Lines.Add('From   ');   //Во вторую строку пишем From
 Memo1.Lines.Add(';'); // В третья строку ";"
 // Установка курсора в позицию 0,8
 p.X:=8;  p.Y:=0;  //  координаты для курсора в Memo
 Memo1.CaretPos:=p; // Выбор позиции 8,0
 Memo1.SetFocus;   // Установка курсора. Заготовка готова.
 // Очищаем сетку DBGrid2 от старого запроса
 dm.ADOQuery2.SQL.Text:='';
end;
```

4.5 На компоненте **Memo1** устанавливаем свойство ScrollBars=ssBoth (добавляем горизонтальную и вертикальную полосы прокрутки).

4.6 Сохраняем проект (**File -> Save All)** и компилируем (**F9).**

!!! Получился неплохой тренажер для изучения языка SQL. Испытаем?

4 Краткая теория по созданию SQL-запросов

Для выборки данных используется оператор **SELECT**. Сокращенный формат этого оператора имеет следующий вид:

SELECT Список_Полей /* Обязательное поле */

FROM Список_Таблиц /* Обязательное поле */

WHERE Критерий выбора записей /*Не обязательное поле */

ORDER BY Список полей /*Не обязательное поле */

где

ORDER BY – параметр, который задает условие упорядоченности записей, удовлетворяющих критерию запроса. По умолчанию – сортировка по возрастанию, а с ключевым словом DESC – по убыванию.

Некоторые ограничения:

1. Точка с запятой (;) – завершающий элемент SQL-запроса.
2. Перечисление полей (таблиц) осуществляется через запятую.
3. Для имен (полей или таблиц) с пробелами или дефисами следует использовать квадратные скобки. Например, [Код товара], [Фамилия сотрудника базы], [Кол-во], [Апрель-Август].

4. Для одинаковых имен полей в таблицах, используемых в запросе, следует перед их названием писать имя таблицы. Например, Заказы.КодТовара и Товары.КодТовара.
5. Строки обязательно записываются в кавычках. Например, Клиенты.ФИО = 'Иванов И.П.'.

В нижеприведенных примерах используется БД Товар, созданная в Лабораторной работе №11. Исполнение примеров осуществляется в разработанной Информационной системе «Товар».

Пример 1. Вывести записи всех полей таблицы **Клиенты.**

Текст клиентского SQL-запроса (Вводить в ручную)

```
Select *
From Клиенты
;
```

Исполнение клиентского SQL-запроса

Код	ФИО	Адрес	Телефон
1	Иванов В.И.	Степная 15	896735655661
2	Дьяченко И.В.	Ленина 5	896611111111
3	Сидокенко И.А.	Солнечный 12	896522222222
4	Васильев Г.В.	Тухачевского 37/1 кв 5	
5	Илларионов М.П.	Кулакова 295/1 кв 15	

Пояснение: вместо перечисления имен всех полей можно использовать символ «*» (звездочка).

Пример 2. Вывести записи всех полей таблицы **Товары**, упорядоченных по возрастанию стоимости закупки.

Текст клиентского SQL-запроса (Вводить в ручную)

```
Select * From Товары
Order by СтоимостьЗакупки ;
```

В случае требования упорядочивания по убыванию стоимости закупки:

Текст клиентского SQL-запроса (Вводить в ручную)

```
Select * From Товары
Order by СтоимостьЗакупки DESC;
```

Пример 3. Вывести записи полей **КодТовара, НаименованиеТовара, СтоимостьЗакупки** таблицы **Товары**, где стоимость закупки товаров колеблется от 1000 до 4000 у.е. Произвести упорядочение по возрастанию стоимости закупки.

Текст клиентского SQL-запроса (Вводить в ручную)

```
Select  КодТовара, НаименованиеТовара, СтоимостьЗакупки
From  Товары
Where  (СтоимостьЗакупки>1000) AND (СтоимостьЗакупки<4000)
Order by СтоимостьЗакупки DESC;
```

Выполнить запрос Создать новый SQL-запрос

Исполнение клиентского SQL-запроса

КодТовара	НаименованиеТовара	СтоимостьЗакупки
11	Колонки MICROLAB H510 (5.1)	3150
9	Модем D-Link	1990
12	Стол д/комп DL-004/beech	1980
8	Модем Acorp	1450

Пример 4. Вывести записи полей **КодТовара, НаименованиеТовара, СтоимостьЗакупки, Наличие, Кол-во** таблицы **Товары**. В запрос включить только имеющиеся товары. Упорядочить по полю **Кол-во**.

Текст клиентского SQL-запроса (Вводить в ручную)

```
SELECT Товары.КодТовара, Товары.НаименованиеТовара, Товары.СтоимостьЗакупки,
Товары.Наличие, Товары.[Кол-во]
FROM Товары
WHERE (Товары.Наличие=True)
ORDER BY Товары.[Кол-во] ;
```

Выполнить запрос Создать новый SQL-запрос

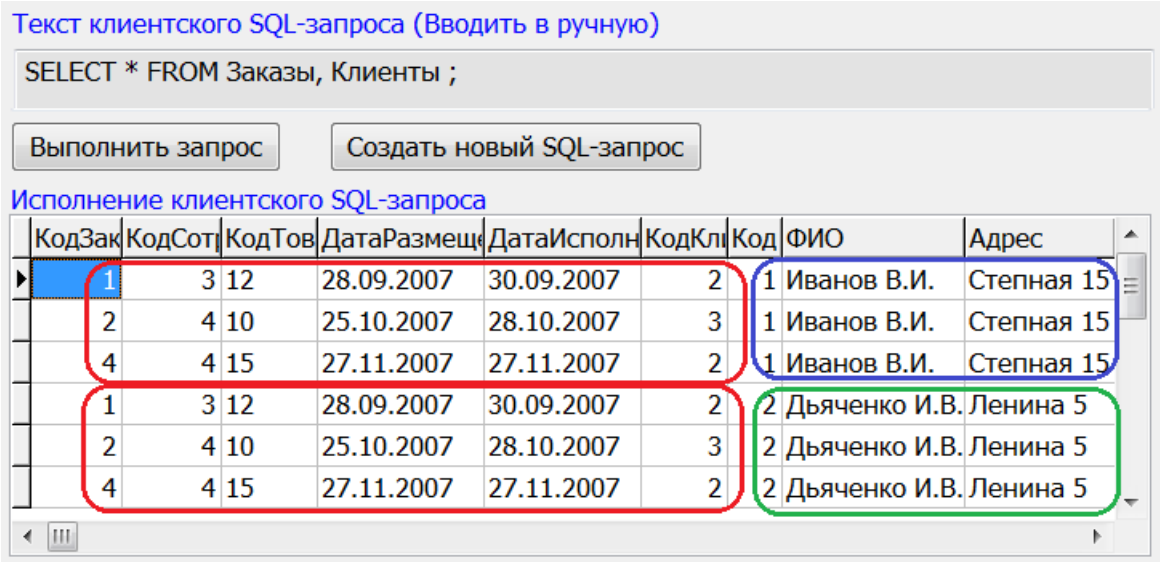
Исполнение клиентского SQL-запроса

КодТова	НаименованиеТовара	СтоимостьЗакупки	Наличие	Кол-во
10	Сканер Perfection V700 Photo	12560	True	1
2	Ноутбук MSI	30600	True	3
18	Беспроводная мышь Genius	385	True	3
9	Модем D-Link	1990	True	4
7	Компьютер FORUM Intel Core 2 Duo	24100	True	5
16	Компьютер Dell Dimension 8300	35100	True	10
1	Ноутбук Acer222	34000	True	12

Замечание. Порядок вывода полей в запросе указан в операторе **Select**.

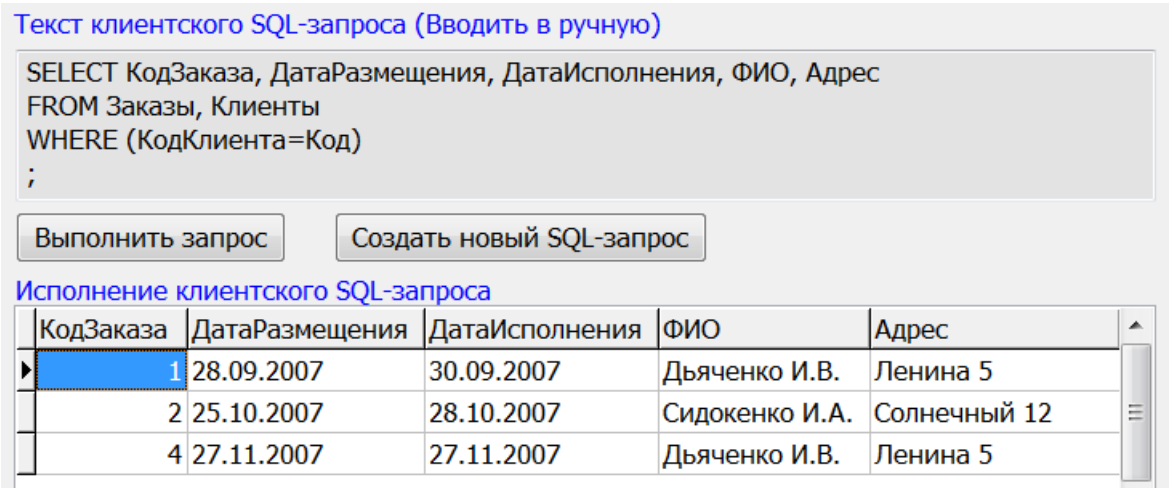
Пояснение. В учебных целях, использованы полные имена полей (включая имя таблицы). Квадратные скобки в названии Товары.[Кол-во] обязательны, т.к. дефис воспринимается также как пробел.

Пример 5. Бессмысленное объединение двух таблиц, запрос соединяет каждую строку из таблицы **Заказы** с каждой строкой из таблицы **Клиенты** в одну строку (т.е. декартовое произведение).



КодЗак	КодСот	КодТов	ДатаРазмеще	ДатаИсполн	КодКл	Код	ФИО	Адрес
1	3	12	28.09.2007	30.09.2007	2	1	Иванов В.И.	Степная 15
2	4	10	25.10.2007	28.10.2007	3	1	Иванов В.И.	Степная 15
4	4	15	27.11.2007	27.11.2007	2	1	Иванов В.И.	Степная 15
1	3	12	28.09.2007	30.09.2007	2	2	Дьяченко И.В.	Ленина 5
2	4	10	25.10.2007	28.10.2007	3	2	Дьяченко И.В.	Ленина 5
4	4	15	27.11.2007	27.11.2007	2	2	Дьяченко И.В.	Ленина 5

Пример 6. Объединение 2-х таблиц с помощью слова **WHERE**. Организовать выборку всех заказов (в виде новой таблицы), указав в ней Код заказа, Дату размещения, Дату исполнения (из таблицы **Заказы**), ФИО и адрес заказчика (из таблицы **Клиенты**).



КодЗаказа	ДатаРазмещения	ДатаИсполнения	ФИО	Адрес
1	28.09.2007	30.09.2007	Дьяченко И.В.	Ленина 5
2	25.10.2007	28.10.2007	Сидоренко И.А.	Солнечный 12
4	27.11.2007	27.11.2007	Дьяченко И.В.	Ленина 5

Пояснение. В таблице **Заказы** всего 3 записи. Все они выведены. В описание имен полей, использовать имена таблиц не обязательно.

Пример 7. Объединение 3-х таблиц с помощью слова **WHERE**. Организовать выборку всех заказов (в виде новой таблицы), указав в ней Наименование товара (из таблицы **Товары**), Дату размещения, Дату исполнения (из таблицы **Заказы**), ФИО и адрес заказчика (из таблицы **Клиенты**). Упорядочить по убыванию Даты исполнения.

Текст клиентского SQL-запроса (Вводить в ручную)

```
SELECT НаименованиеТовара, ДатаРазмещения, ДатаИсполнения, ФИО, Адрес
FROM Товары, Клиенты, Заказы
WHERE (Код=КодКлиента) AND (Товары.КодТовара = Заказы.КодТовара)
ORDER BY ДатаИсполнения DESC
;
```

Выполнить запрос | Создать новый SQL-запрос

Исполнение клиентского SQL-запроса

НаименованиеТовара	ДатаРазмещ	ДатаИсполн	ФИО	Адрес
КПК	27.11.2007	27.11.2007	Дьяченко И.В.	Ленина 5
Сканер Perfection V700 Photo	25.10.2007	28.10.2007	Сидоренко И.А.	Солнечный 12
Стол д/комп DL-004/beech	28.09.2007	30.09.2007	Дьяченко И.В.	Ленина 5

Пример 8. Объединение 3-х таблиц с помощью операции **INNER JOIN** (пересечение). Организовать выборку всех заказов (в виде новой таблицы), указав в ней Наименование товара (из таблицы **Товары**), Дату размещения, Дату исполнения (из таблицы **Заказы**), ФИО и адреса заказчика (из таблицы **Клиенты**). Упорядочить по убыванию Даты исполнения.

Текст клиентского SQL-запроса (Вводить в ручную)

```
SELECT НаименованиеТовара, ДатаРазмещения, ДатаИсполнения, ФИО, Адрес
FROM Товары INNER JOIN (Клиенты INNER JOIN Заказы ON Клиенты.Код =
Заказы.КодКлиента) ON Товары.КодТовара = Заказы.КодТовара
ORDER BY ДатаИсполнения DESC
;
```

Выполнить запрос | Создать новый SQL-запрос

Исполнение клиентского SQL-запроса

НаименованиеТовара	ДатаРазмещ	ДатаИсполн	ФИО	Адрес
КПК	27.11.2007	27.11.2007	Дьяченко И.В.	Ленина 5
Сканер Perfection V700 Photo	25.10.2007	28.10.2007	Сидоренко И.А.	Солнечный 12
Стол д/комп DL-004/beech	28.09.2007	30.09.2007	Дьяченко И.В.	Ленина 5

Замечание. Объединение **INNER JOIN** всегда можно заменить объединением **WHERE** (сравнить результаты примеров 7 и8).

Пояснение. В результат объединения **INNER JOIN** попадают те записи, которые строго удовлетворяют условию объединения, которое указывается после слова **ON**. Сначала формируется множество записей объединения таблиц **Клиенты и Заказы**, т.е. выполняется скобка (Клиенты **INNER JOIN** Заказы **ON** Клиенты.Код = Заказы.КодКлиента). Затем происходит объединение с таблицей **Товары**, по ключевым полям КодТовара.

Пример 9. Объединение 2-х таблиц с помощью операции **LEFT JOIN** (пересечение с необязательным присутствием слева).

Организовать выборку всех товаров (в виде новой таблицы), указав в ней Наименование товара, Количество (из таблицы **Товары**), а также, если были заказы, то добавить Код заказа, Дату размещения и Дату исполнения заказа (из таблицы **Заказы**). В выборку включать только имеющиеся товары.

Текст клиентского SQL-запроса (Вводить в ручную)

```
SELECT НаименованиеТовара, [Кол-во], КодЗаказа, ДатаРазмещения, ДатаИсполнения
FROM Товары LEFT JOIN Заказы ON Товары.КодТовара = Заказы.КодТовара
WHERE Наличие=True
;
```

Выполнить запрос | Создать новый SQL-запрос

Исполнение клиентского SQL-запроса

НаименованиеТовара	Кол-во	КодЗаказа	ДатаРазмещения	ДатаИсполнения
Ноутбук Acer222	12			
Сканер Perfection V700 Photo	1	2	25.10.2007	28.10.2007
Стол д/комп DL-004/beech	25	1	28.09.2007	30.09.2007
Лампы для подсветки	50			
КПК	25	4	27.11.2007	27.11.2007
Компьютер Dell Dimension 8300	10			

Пояснение. В запрос выбираются все строки из таблицы **Товар** (слева от **LEFT JOIN**). К ним добавляются строки из таблицы **Заказы** (справа от **LEFT JOIN**), причем выбирает только те пары, которые соответствуют выражению указанному после слова **ON**. Если для какой-то строки из таблицы **Товар** не нашлось ни одной строки из таблицы **Заказы,** соответствующей условию, то строка соединяется с полями, имеющими значения NULL (нет значения). К этим записям применяется операция наличия: **WHERE Наличие=True**.

Пример 10. Работа с датами и текстом. Организовать выборку Модемов, указав в ней Наименование товара, Стоимость закупки (из таблицы **Товары**), которые поступили с сентября 2007 (из таблицы **Поставка**).

Текст клиентского SQL-запроса (Вводить в ручную)

```
SELECT НаименованиеТовара, СтоимостьЗакупки, ДатаПоставки
FROM Поставка INNER JOIN Товары ON Поставка.КодПоставки = Товары.КодПоставки
WHERE (НаименованиеТовара LIKE 'Модем%') AND (ДатаПоставки>=#9/1/2007#) ;
```

Выполнить запрос | Создать новый SQL-запрос

Исполнение клиентского SQL-запроса

НаименованиеТовара	СтоимостьЗакупки	ДатаПоставки
Модем Acorp	1450	28.09.2007
Модем D-Link	1990	28.09.2007

Пояснение. Подстановочный символ «%» замещает любое количество символов. Операция сравнения **LIKE** 'Модем%' – отбирает записи, которые

начинаются со слова «Модем». Искомый текст заключен в одинарные кавычки ('). Дата приведена в формате #Месяц/День/Год#, что верно для MS Access. В других системах формат написания даты может быть другим.

6 Задания для самостоятельной работы

6. Усовершенствуйте проект своего варианта задания, организуя выбор информации из БД с помощью заданного и клиентского SQL-запросов, по аналогии с разработкой Form5 проекта «Информационная система Товар». Создайте пять SQL-запросов для своей базы данных.

Учебное издание

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ
по дисциплине
«УПРАВЛЕНИЕ ИНФОРМАЦИЕЙ И ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ»
для студентов направления подготовки
Профессиональное обучение (по отраслям),
профиль «Информационные технологии и системы»
(в 2-х частях). Часть 1.

С о с т а в и т е л ь:
Елена Сергеевна Чёрная

Печатается в авторской редакции.
Компьютерная верстка и оригинал-макет автора.

Подписано в печать _____
Формат 60х84$^{1}/_{16}$. Бумага типограф. Гарнитура Times
Печать офсетная. Усл. печ. л. _____. Уч.-изд. л. _____
Тираж 100 экз. Изд. №_____. Заказ №_____. Цена договорная.

Издательство Луганского государственного
университета имени Владимира Даля

Свидетельство о государственной регистрации издательства МИ-СРГ ИД 000003 от 20 ноября 2015г.

Адрес издательства: 91034, г. Луганск, кв. Молодежный, 20а
Телефон: 8 (0642) 41-34-12, **факс:** 8 (0642) 41-31-60
E-mail: izdat.lguv.dal@gmail.com **http:** //izdat.dahluniver.ru/