МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра "Интеллектуальные информационные технологии"

Средства обработки баз данных в средах программирования Delphi и C++Builder

Методические указания к выполнению лабораторных работ
по дисциплине **"Базы и банки данных"**
для студентов специальности
1 530102 «Автоматизированные системы обработки информации»

Брест 2007

УДК 681.3

В методических указаниях приведены необходимые теоретические сведения о возможностях использования среды программирования Delphi или C++Builder как системы управления базами данных. Содержатся задания и указания к выполнению пяти лабораторных работ по указанной тематике (постановка задачи, содержание отчета и контрольные вопросы для проверки), а также варианты заданий.

Методические указания предназначены для использования студентами специальности 1 53 01 02 в ходе выполнения лабораторных работ по дисциплине «Банки и базы данных», а также могут быть полезны программистам при создании приложений обработки баз данных.

ОГЛАВЛЕНИЕ

Введение

Методические указания содержат теоретические сведения, задания и примеры программ для обработки баз данных (БД) для сред Delphi и C++Builder, разработанных фирмой Borland. Это визуальные среды объектно-ориентированного программирования, очень удобные в использовании.

Корпорация Borland предлагает разработчикам широкий выбор средств разработки приложений.

Получившие признание решения Borland Delphi и Borland C++Builder для Windows повышают производительность разработчиков. Для высокой производительности разработчика очень важна хорошо продуманная библиотека компонентов. У инструментов Delphi и C++Builder общая библиотека компонентов, поэтому, изучив состав и структуру компонентов, легко использовать любой из программных продуктов. Отличие состоит в языке написания кода: Object Pascal или C++.

Как Borland Delphi, так и Borland C++Builder имеют все необходимые возможности разработки приложений для платформы Windows.

Для создания приложений с помощью Delphi или C++Builder используется технология визуального программирования. В палитре выбираются компоненты, устанавливаются их свойства. Затем пишется бизнес-логика на языке Delphi (на основе ObjectPascal) или C++. С целью оптимизации быстродействия готовая система компилируется собственными средствами.

Это значительно повышает производительность разработки. Компоненты спроектированы так, что в большинстве случаев они обрабатывают действия приложения без дополнительного кода. Код, который пишет разработчик, это, как правило, бизнес-логика, в соответствии с которой конкретная система отвечает на события.

Для работы с БД в Delphi и C++Builder есть несколько наборов компонент. Все они используют разные технологии доступа к данным и отличаются по возможностям. В отличие от фирмы Microsoft, которая встроила в свои продукты только технологию доступа ADO, фирма Borland дала разнообразие средств через разные технологии.

BDE (Borland Database Engine) – для доступа к данным в таблицах стандартного формата Borland: Paradox, dBase.

ADO (Active Data Object)- технология разработана фирмой Microsoft, используется для доступа к БД ACCESS или Microsoft SQL.

DbExpress –новая технология фирмы Borland, подходит для разработки клиент-серверных приложений- Oracle, DB2, MySQL.

Работа с компонентами различных технологий похожа. Поэтому для примера и усвоения основных понятий обработки БД в данном издании рассматривается технология доступа к данным BDE.

Разработчик может выбирать среду программирования: Delphi или C++Builder.

Лабораторная работа № 1

Физическая структура БД. Таблицы, приложения просмотра таблиц Delphi

Цель работы. Получить навыки работы с BDE при создании структуры таблиц и редактирования записей.

Методические рекомендации

Процессор баз данных Borland Database Engine

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

поиск местоположения базы данных;

поиск таблицы, ее открытие и чтение служебной информации;

чтение данных в соответствии с форматом хранения данных.

Все стандартные функции доступа к данным реализуются в виде специальной программы, сервиса или динамической библиотеки.

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

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

Взаимодействие процессора баз данных с различными БД

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

Однако BDE имеет некоторые недостатки. Это, например, снижение скорости работы приложения, недостатки реализации некоторых драйверов и т. д

Архитектура и функции BDE

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

В составе BDE поставляются стандартные драйверы, обеспечивающие доступ к СУБД Paradox, dBASE, FoxPro и текстовым файлам. Локальные драйверы устанавливаются автоматически совместно с ядром процессора

Псевдонимы баз данных и настройка BDE

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

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

Для решения такого рода проблем разработчик может использовать псевдоним базы данных, который представляет собой именованную структуру, содержащую путь к файлам БД и некоторые дополнительные параметры. В первом приближении можно сказать, что вы просто присваиваете маршруту произвольное имя, которое используется в приложении. Тогда при переносе приложения на компьютере заказчика достаточно создать стандартными средствами BDE одноименный псевдоним и настроить его на нужный каталог. При этом приложение не требует редактирования.

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

Для управления псевдонимами баз данных, настройки стандартных и дополнительных драйверов в составе BDE имеется специальная утилита — BDE Administrator (исполняемый файл BDEADMIN.EXE), запускается через Главное меню Windows. Стандартная конфигурация BDE сохраняется в файле IDAPI.CFG.

Для возможности отображения русских символов в таблицах Paradox следует произвести настройку драйвера через BDE Administrator.

Вкладка Configuration- Drivers- Native- Paradox- LangDriver, выбрать драйвер Pdox ANSI Cyrillic. Сохранить конфигурацию. Настройки драйвера применяются при следующем запуске всех приложений, связанных с BDE.

Создание формы

Необходимые компоненты для просмотра данных.

Вкладка	Компоненты	Свойства, требующие идентификации с помощью инспектора объектов
DataAccess	Ttable	DatabaseName TableName Active
	TdataSource	DataSet
DataControls	TDBGrid	DataSource

Другие компоненты

Standart	Tbutton	Name Caption OnClick (обработчик события)

Задание

1. Разработать физическую модель БД для реализации в СУБД DELFI на основании логической модели. Для данной лабораторной работы выбрать три связанные таблицы. Для создания таблиц использовать типы СУБД Paradox, драйвер баз данных которой является стандартным для Delphi.

Пример.

Логическая модель БД

Код товара	Наименование товара	Единица измерения

Код товара	Дата поступления	Количество

Построим физическую модель.

Перед построением физической модели изучить типы полей, поддерживаемых СУБД PARADOX.

Имя Файла TOVAR.DBF

Назначение поля	Имя поля	Тип	Объем
Код товара	KOD	Short	2
Наименование товара	NameT	Alpha	20
Единица измерения	ED	Alpha	2

Имя файла POST_TOVAR.DBF

Назначение поля	Имя поля	Тип	Объем
Код товара	KOD	Short	2
Дата поступления	DPost	Date	8
Количество	Kol	Number	8

Примечание: Ключевое поле выделено ***жирным курсивом***.

Объемы всех типов полей, кроме Alpha, стандартны и изменяться не могут. Для определения количества занимаемых байт следует воспользоваться справочной информацией или расчетами (СПРАВКА Database Desktop-Tables- Creating Tables- defining Fields-Field Types and Sizes).

2. Запустить утилиту BDE Administrator с диска и создать псевдоним своей БД, которая будет находиться в определенном директории.
3. Запустить Delphi, затем утилиту Database Desktop(DBD) через меню Tools.

В DBD настроить рабочий директорий File- Working Directory, указав имя созданного псевдонима.

4. Создать структуру выбранных таблиц, File-New-Table, определить первичные ключи, заполнить таблицы несколькими записями, сохранить данные.
5. Изменить структуру одной из таблиц:

   File-Open-Table - открыть,

   Table-Restructure - изменить структуру.
6. Определить следующие свойства полей: min величина, max величина, значение по умолчанию, обязательность заполнения поля. Для одного поля задать шаблон (например, для ввода телефона шаблон будет выглядеть ЦИФРА ЦИФРА- ЦИФРА ЦИФРА-ЦИФРА ЦИФРА, т.е. ##-##-##)
7. Определить вид целостности данных между таблицами:

   в режиме изменения структуры Table Properties - Refrential Integrity - Define.

   Выбрать вид каскадных воздействий на дочернюю таблицу. Задать имя связи.

   Закрыть DBD.
8. Создать в своем директории подкаталог для хранения приложений.

Создать одну форму по описанной ниже технологии для просмотра данных первой и второй таблиц. Использовать компоненты TdataSource, Ttable, TDBGrid. Сохранить и запустить приложение. Попробовать добавлять, удалять, редактировать данные.

В чем проявляется поддержание целостности данных при редактировании?

9. Организовать связь со второй таблицей типа Master-Detail (родительская- дочерняя). (Использовать компонент Ttable, свойства Master Source, Master Fields). В диалоговом окне в поле Available Indexes выбрать индекс по полю связи. Запустить приложение. В чем проявляется связь при просмотре?
10. Открыв таблицу в режиме структуры, создать индексный файл по любому полю (неключевому) - поле ввода Secondary Indexes. Установить этот индекс активным (свойство IndexName компоненты TTable). Запустить приложение. Как изменился порядок расположения записей в таблице? В каком порядке записи находятся изначально?

Какой индексный файл был создан? Где он располагается?

Чтобы набор данных нельзя было переводить в состояние добавления и удаления данных из компоненты DBGrid, установить свойство DBGrid.ReadOnly=True.

Содержание отчета

1. Тема и цель работы.
2. Физическая структура таблиц БД.
3. Имена и типы связей целостности данных и Master-Detail, имена и поля индексных файлов.
4. Распечатка формы результата (по возможности).
5. Пример попытки нарушения целостности при добавлении конкретных данных
6. Пример просмотра данных второй таблицы со связью и без (значение выбранной текущей записи).

Контрольные вопросы

1. Что такое псевдоним БД и как он создается?
2. Критерии определения степени ассоциативности связей между таблицами.
3. Критерии определения главной и зависимой таблицы для связи.
4. Какие ошибки могут возникнуть при добавлении данных? (Key violation, Master Record Missing).
5. Отличие связи целостности данных от связи для просмотра данных.
6. Назначение утилиты *Database Desktop*.
7. Назначение и активизация индексного файла.
8. В каких случаях индексные файлы создаются автоматически?

Лабораторная работа № 2

Организация интерфейса

Цель работы. Получить навыки использования компонент Delphi для организации интерфейса.

Методические рекомендации

Визуальные компоненты для работы с текущей записью набора данных

TDBText – *аналог TLabel*

Отображение значения текстового поля текущей записи

Свойства: DataSource, DataField

TDBEdit – TEdit

Редактирование значения строкового поля текущей записи.

Свойства: DataSource, DataField, ReadOnly.

TDBCheckBox – TcheckBox

Позволяет установить значение логического поля текущей записи.

Свойства: DataSource, DataField, ReadOnly.

Свойство Checked имеет значение True, если поле отмечено, и False иначе.

Свойство State возвращает текущее состояние поля. Существуют следующие константы значения поля:

CbUnchecked, cbChecked, cbGrayed (пустое значение).

TDBRadioGroup - TradioGroup

Предоставляет выбор одного из группы возможных значений поля.

Свойства: DataSource, DataField, ReadOnly.

Свойство Columns указывает, сколько столбцов выводится на экран в компоненте.

Событие OnChange наступает при изменении значения поля.

События OnEnter, OnExit при получении и утрате фокуса компонентой соответственно.

TDBListBox - TListBox

Выбор значения поля из списка.

Свойства: DataSource, DataField, ReadOnly.

Свойство Item содержит список возможных значений поля, к конкретному значению можно обратиться по его номеру, начиная с 0.

События OnEnter, OnExit при получении и утрате фокуса компонентой соответственно.

События OnClick, OnDblClick наступают при одиночном и двойном щелчке на компоненте соответственно.

TDBLookupComboBox - TLookupComboBox

Выбор значения поля из списка, содержащегося в другой таблице (обычно родительской).

Свойства:

ListSource- источник данных для создания списка (родительская таблица)- откуда

ListField- поле для списка

DataSource – источник данных для внесения значений -куда

DataField –поле для внесения

KeyField- из какого поля источника ListSource вносить данные в другую таблицу.

ВСЕ 5 свойств должны быть заданы!

Возможен вариант нетождественности полей KeyField и ListField. Пример:

ListSource

Код товара	Наименование	Цена

DataSource

Нечека	Код_тов	Количество

ListField- Наименование
DataField –Код_тов
KeyField-Код товара.
В этом случае в раскрывающемся списке будут отражаться наименования товаров, а записываться в другую таблицу будет Код товара.

Невизуальные компоненты для организации интерфейса
Компонент TmainMenu определяет главное меню формы.
Меню отображается в верхней полосе формы для объекта, заданного в свойстве Menu формы.
Список значений меню содержится в свойстве Item.
Подменю можно создать, используя пункт контекстного меню Create Submenu.

Способы доступа к полям компоненты Ttable.
Получить доступ к значению поля таблицы можно через метод FieldByName компоненты Ttable. Параметром этого метода служит ссылка на конкретное поле по имени поля. Для данного метода должен быть указан тип преобразования значения поля.

Пример:
```
table1.FieldByName('fio').AsString
Table1->FieldByName("Continent")->AsString;//C++
```

Второй способ доступа к данным таблицы – использование семейства Fields, которое включает в себя совокупность всех полей таблицы. К полю можно обратиться по имени, по номеру, через свойство Item. Пример:
```
Table1.Fields[0].Value:='Иванов' ;//DELPHI
Table1->Fields->Fields[2]->AsString ="Иванов";//C++
Table1->Fields->Fields[2]->Value ="Иванов";//C++
```

Третий способ – использование свойства FieldValues:
```
Table1.FieldValues['fio']:='Иванов'; //DELPHI
Table1->FieldValues["fio"]="Иванов"; //C++
```

Изменения в таблице
Для запоминания изменений используется метод Post, для отмены изменений – метод Cancel компоненты Ttable. Пример:
```
//DELPHI
procedure    TForm1.FormClose(Sender:    TObject;    var    Action:
TCloseAction)      ;
begin
table1.post
end;
procedure TForm1.FormActivate(Sender: TObject);
begin
    edit1.Text:=table1.FieldByName('fio').AsString;
end;

//C++
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction
&Action)
{
Table1->Post();
}
//C++
void __fastcall TForm1::FormActivate(TObject *Sender)
{
Edit1->Text=Table1->FieldByName("Capital")->AsString;
}
```

Методы компонента TTable:

Delete - удаление текущей записи;

Insert- добавление записи в конец таблицы;

Post- сохранение записей.

ВНИМАНИЕ! Перед применением метода POST обязательно должен быть применен метод EDIT или INSERT.

First, Last, Next, Pred - перемещение текущего указателя по записям таблицы соответственно.

Методы компонента TForm:

Show – открытие формы;

ShowModal – открытие формы в модальном (преимущественном) режиме;

Close- закрытие формы.

События компонента TForm: OnActivate, OnClose.

События компонента TMainMenu: OnClick.

События компонента TButton: OnClick.

Задание

Таблицы для всех лабораторных работ выбираются в соответствии с вариантом задания.

1. Выбрать дочернюю таблицу. Организовать на главной форме просмотр выбранной таблицы.
2. Создать на главной форме меню вида ОТКРЫТЬ ТАБЛ, ВВОД, Диаграмма В пункте меню ВВОД создать подменю ВВОД GRID, ВВОД DB.
3. Создать 2 формы для ввода данных в одну и ту же таблицу, используя все перечисленные компоненты (самостоятельно выбрать, для каких полей их удобней использовать). Дать мнемонические имена переменных для всех объектов (например, по умолчанию свойство Name= Table1 для компонента Ttable; изменить его на Student). Это задание необходимо выполнить для того, чтобы текст программы был уникальным для каждой БД.

   1 форма - DBGrid, DBNavigator.

   2 форма– DBEdit, DBComboBOx (или DBListBox), DBLookupComboBox

   Обеспечить целостность данных при вводе значения связанного поля, в учебных целях в 2-х вариантах. Для этого использовать для ввода значения поля связи 2 варианта на одной форме: список с заданными значениями (DBComboBOx или DBListBox) и список для выбора значений из родительской таблицы DBLockupComboBox.

   Добавить подписи к вводимым данным, используя компоненты Tlabel.

   Назначить открытие форм и таблицы соответствующим пунктам меню.
4. В соответствии с хорошим стилем программирования добавить процедуры обработки события Открытия и Закрытия формы. При открытии формы открывать таблицу, при закрытии формы закрывать таблицу. Для формы ВВОД DB, кроме того, при открытии формы добавлять в таблицу новую пустую запись.
5. Проверить, сохраняются ли данные в каждом из 2 случаев в таблице, запустив приложение. Отметить, для каких форм следует добавить сохранение данных.
6. Добавить на каждой форме ввода кнопки ОК и Отмена. Создать процедуры обработки событий нажатия кнопок- по кнопке ОК следует сохранить внесенную запись в таблицу, по кнопке Отмена просто закрыть форму.

7. Обеспечить целостность данных при вводе числового ключа (если ключ строковый, то любого числового поля или даты). Для этого вместо поля ввода DBEDIT добавить метку для вывода данных из ключевого поля DBText. В процедуре обработки события Открытия формы ВВОД DB добавить расчет значения поля по следующему алгоритму:
   --после метода Insert - переместить указатель в таблице на предпоследнюю запись;
   -- прочитать в переменную значение поля любым методом, приведенным для доступа к данным, используя имя поля;
   --увеличить значение переменной на 1;
   --переместить указатель на следующую запись;
   --присвоить полю новое значение переменной.
8. Добавить на главную форму компонент DBChart и построить диаграмму зависимости числового поля от любого другого. Установить свойство Visible в false. Создать процедуру обработки события нажатия пункта меню Диаграмма: сделать невидимыми визуальные компоненты главной формы
   - Сделать видимым компонент для отображения диаграммы.
9. Организовать интерфейс по своему выбору для отображения таблицы вместо диаграммы на главной форме и для удаления записи из таблицы.

Содержание отчета

Цель работы
По каждому пункту задания:
   Имена объектных переменных (формы, таблицы, источники данных и т.д.)
   Текст процедур обработки событий.

Контрольные вопросы

1. В чем отличие компонент Edit и DBEdit?
2. В чем отличие компонент DBComboBox и DBLookupComboBox?
3. При использовании каких компонент для изменения данных таблицы требуется и не требуется дополнительное сохранения данных методом POST?
4. Способы доступа к полям таблицы.
5. В чем отличие методов для открытия формы Show и ShowModal?
6. Почему хороший стиль программирования требует открытия и закрытия таблицы для каждой формы?

Лабораторная работа №3

Поиск и отбор данных

Методические рекомендации

Поиск записей в наборах данных (НД)

Метод Locate

```
//DELPHI
function Locate(const KeyFields: string; const KeyValues: Variant;
Options: TLocateOptions): Boolean;

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

Метод *Locate* ищет первую запись, удовлетворяющую критерию поиска, и если такая запись найдена, делает ее текущей. В этом случае в качестве результата возвращается True. Если поиск был неуспешен, возвращается False.

Параметры:

Список *KeyFields* указывает поле или несколько полей, по которым ведется поиск, в виде строкового выражения. В случае нескольких поисковых полей их названия разделяются точкой с запятой.

Критерии поиска задаются в вариантном массиве *KeyValues* так, что i-е значение в *KeyValues* ставится в соответствие i-му полю в *KeyFields*. В случае поиска по одному полю в *KeyValues* указывается одно значение.

Options позволяет указать необязательные значения режимов поиска:

loCaseInsensitive - поиск ведется без учета высоты букв, т.е. если в *KeyValues* указано 'принтер', а в некоторой записи в данном поле встретилось 'Принтер' или 'ПРИНТЕР', запись считается удовлетворяющей условию поиска;

loPartialKey - запись считается удовлетворяющей условию поиска, если она содержит часть поискового контекста; например, удовлетворяющими контексту 'Ма' будут признаны записи с значениями в искомом поле "Машин', 'Макаров' и т.д.

Locate отличается от методов *FindKey, FindNearest, GoToKey, GoToNearest* (компонент TTable) следующим:

FindKey, FindNearest, GoToKey, GoToNearest производят поиск только по полям, входящим в состав текущего индекса TTable; в случае, когда условию поиска удовлетворяет несколько записей, текущей станет логически самая первая из них (в порядке сортировки записей в НД, определяемом текущим индексом);

Locate производит поиск по любому полю; поле или поля, по которым производится поиск, могут не только не входить в текущий индекс, но и не быть индексными вообще. В случае, если поля поиска входят в какой-либо индекс, *Locate* использует этот индекс при поиске. Если искомые поля входят в несколько индексов, трудно сказать, какой из них будет использован. Соответственно, трудно предсказать, какая запись из множества записей, удовлетворяющих критерию поиска, будет сделана текущей - особенно в случае, если поиск ведется не по текущему индексу. При поиске по полям, не входящим ни в один индекс, применяются фильтры BDE.

Пример А. Пусть имеется ТБД "Сотрудники кафедры" с целочисленным TabNum (табельный номер) и строковыми полями FIO (ФИО), Dolj̆nost (Должность), UchStepen (Учёная степень).

Пусть ТБД имеет индексы по полям:

'TabNum'

'FIO'

'Doljnost;FIO'

Осуществим поиск доцента, кхн. Поисковый контекст – ['доцент','кхн'] при режиме частичного совпадения значений:

```
//DELPHI
procedure TFormX.LocateButtonClick(Sender: Tobject);
begin
Table1.Locate('Doljnost;UchStepen',VarArrayOf(['доцент','кхн']),
[loPartialKey]);
end;

//C++
void __fastcall TForm1::LocateButtonClick(TObject *Sender)
{TLocateOptions Op;
 Variant my[2];
my[0] = Variant("доцент");
my[1] = Variant("кхн");
Op.Clear();
Op << loPartialKey;
Table1->Locate("Doljnost;UchStepen",VarArrayOf(my,1),Op);}
```

Пример В. Пусть заранее неизвестно, по какому полю необходимо производить поиск. Тогда поместим в форму компонент RadioGroup1, в котором перечислим поля поиска, и компонент Edit1 для ввода условий поиска:

Напишем такой обработчик:

```
//DELPHI
procedure TFormX.Button1Click(Sender: TObject);
var Pole : Shortstring;
begin
CASE RadioGroup1.ItemIndex OF
0 : Pole := 'TabNum';
1 : Pole := 'FIO';
2 : Pole := 'Doljnost';
3 : Pole := 'UchStepen';
END;
IF not Table1.Locate(Pole,Edit1.Text,[loCaseInsensitive, loPar-
tialKey]) THEN
ShowMessage('Запись не найдена');
end;

//C++
void __fastcall TForm1::Button1Click(TObject *Sender)
{ShortString Pole;
  TLocateOptions Op;
Op.Clear();
Op << loPartialKey,loCaseInsensitive;
switch (RadioGroup1->ItemIndex) {
case 0:Pole="TabNum"; break;
case 1:Pole="FIO"; break;
}
 if (! Table1->Locate(Pole,Edit1->Text,Op))
 ShowMessage("Запись не найдена");
}
```

Преимущество показанного способа в том, что мы вместо выполнения нескольких Locate (для поиска по каждому полю) выполняем один метод Locate независимо от поля, по которому производится поиск.

Метод Lookup

```
//DELPHI
function Lookup(const KeyFields: string; const KeyValues: Variant;
const ResultFields: string):Variant;

//C++
virtual System::Variant __fastcall Lookup(const AnsiString Key-
Fields, const Variant &KeyValues, const AnsiString ResultFields);/C++
```

Метод *Lookup* находит запись, удовлетворяющую условию, но не делает ее текущей, а возвращает значения некоторых полей этой записи. Тип результата - *Variant* или вариантный массив. Независимо от успеха поиска записи, указатель текущей записи в НД не изменяется.

Lookup осуществляет поиск только на *точное* соответствие критерия поиска и значения полей записи. Такой режим, как *loPartialKey* метода *Locate* (поиск по частичному соответствию значений), отсутствует.

Параметры:

В *KeyFields* указывается список полей, по которым необходимо осуществить поиск. При наличии в этом списке более чем одного поля, соседние поля разделяются точкой с запятой.

KeyValues указывает поисковые значения полей, список которых содержится в *KeyFields*. Если имеется несколько поисковых полей каждому i-му полю в списке *KeyFields* ставится в соответствие i-ое значение в списке *KeyValues*. При наличии одного поля, его поисковое значение можно указывать в качестве *KeyValues* непосредственно; в случае нескольких полей - их необходимо приводить к типу вариантного массива при помощи *VarArrayOf*.

В качестве поисковых полей можно указывать поля как входящие в какой-либо индекс, так и не входящие в него; тип текущего индекса не имеет значения. Если поисковые поля входят в какие-либо индексы, их использование производится автоматически; в противном случае используются фильтры BDE.

Если запись в результате поиска не найдена, метод *Lookup* возвращает *Null*, что выявляется при помощи предложения

IF VarType(LookupResults) = varNull THEN ...

В противном случае *Lookup* возвращает из этой записи значения полей, список которых указан в *ResultFields*. При этом размерность результата зависит от того, сколько результирующих полей указано в *ResultFields*:

указано одно поле - результатом будет значение соответствующего типа или *Null*, если поле в найденной записи содержит пустое значение;

указано несколько полей - результатом будет вариантный массив, число элементов в котором меньше или равно числу результирующих полей; меньше потому, что некоторые поля найденной записи могут содержать пустые значения.

Рассмотрим несколько вариантов.

Пример А. Одно результирующее поле (результат - значение типа *Variant*) Будем осуществлять поиск в ТБД "Сотрудники" по полю 'ФИО'. Поисковое значение будем вводить в Edit1. В качестве результата будем выдавать значение поля "UchStepen' (ученая степень) найденной записи.

```
//DELPHI
procedure TFormX.LookuplButtonClick(Sender: TObject);
var
LookupResults : Variant;   //  результат
begin
// осуществить поиск
LookupResults := Table1.Lookup('FIO', Edit1.Text,'UchStepen');
Label1.Caption :='';
```

```
// содержит ли результат пустое значение или Null?
CASE VarType(LookupResults) OF
varEmpty : Label.Caption :='Пустой результат';
varNull : Label1.Caption :='Запись не найдена';
ELSE
// нет, результат содержит какое-то значение
Label1.Caption := LookupResults;
END; //case
end;
```

```
//C++
void __fastcall TForm1::Button1Click(TObject *Sender)
{Variant L;
 L = Table1->Lookup("Capital", Edit1->Text,"Name");
 Label1->Caption ="";
 // содержит ли результат пустое значение или Null?
 switch (VarType(L)) {
 case varEmpty : Label1->Caption ="Пустой результат";break;
 case varNull : Label1->Caption ="Запись не найдена";break;
 default:
 // нет, результат содержит какое-то значение
 Label1->Caption = L;
 } //Switch
```

Заметим, что в присваивании
Label1.Caption := LookupResults;
имеет место приведение вариантного типа к строковому. Более подробно о приведении вариантных типов см. описание вариантного типа в документации и встроенной системе помощи Delphi.

Пример Б. Несколько результирующих полей (результат - вариантный массив)
Некоторые сведения по использованию вариантного массива:
Если переменная типа *Variant* является вариантным массивом, функция *VarIsArray(LookupResults)* возвращает *True*.
При работе с переменным числом возвращаемых полей, в конкретном случае верхнюю и нижнюю границы массива LookupResults можно определить при помощи функций *VarArrayLowBound(LookupResults,1)* и *VarArrayHighBound(LookupResults,1)*.
Тип i-го элемента вариантного массива можно определить как VarType (LookupResults[i]).
Будем осуществлять поиск в ТБД "Сотрудники" по полю 'FIO'. Поисковое значение будем вводить в Edit1. В качестве результата будем выдавать значения полей 'TabNum;Doljnost;UchStepen' найденной записи (табельный номер, должность, ученая степень).

```
//DELPHI
procedure TFormX.LookupButtonClick (Sender: TObject);
var
LookupResults : Variant; // результат
begin
// осуществить поиск
LookupResults := Table1.Lookup ('FIO', Edit1.Text, 'TabNum;Doljnost;UchStepen');
//будем показывать значения результирующих полей в Tlabel
Label1.Caption :='';
Label2.Caption :='';
Label3.Caption :='';
// результат - вариантный массив ?
IF VarIsArray(LookupResults) THEN
begin
```

```
Label1.Caption := LookupResults[0];
IF LookupResults[1] <> Null THEN
Label2.Caption := LookupResults[1];
IF LookupResults[2] <> Null THEN
Label3.Caption := LookupResults[2];
end //then
ELSE
// результат - не вариантный массив, а единичное значение
CASE VarType(LookupResults) OF
varEmpty : Label1.Caption := 'Пустой результат';
varNull : Label1.Caption := 'Запись не найдена';
END; //case
end;
```

Если запись не найдена, *VarType(LookupResults)* возвращает значение *varNull;* если поиск по какой-либо причине не был произведен, *VarType(LookupResults)* возвращает значение *varEmpty.* Если какое-либо из полей, чьи значения возвращаются в результате поиска в вариантном массиве, содержит пустое значение, соответствующий элемент вариантного массива также будет содержать пустое значение *(Null).* В этом случае обращение к нему возбудит исключительную ситуацию, поэтому нужна предварительная проверка.

Фильтрация записей в наборах данных

Помимо описываемых ниже средств, для фильтрации данных могут использоваться:
методы *SetRange или ApplyRange* (и сопутствующие им методы) - в компоненте Table;
секция WHERE оператора SELECT языка SQL - в компоненте Tquery (тема лабораторной работы №5).

Свойство Filtered

property Filtered: Boolean;

Свойство *Filtered,* установленное в True, инициирует фильтрацию, условие которой записано или в обработчике события *OnFilterRecord,* или содержится как строковое значение в свойстве *Filter.* Если установлены разные условия фильтрации и в событии *OnFilterRecord,* и в свойстве *Filter,* выполняются оба.

Например, если в НД одновременно установлены фильтры

```
Table1.Filter = ' [Doljnost] = ''профессор''';
```

// формируем строку [Doljnost]='профессор', которая тоже должна заключаться в кавычки , т.к. свойство Filter имеет тип строка-

```
//DELPHI
  procedure TFormX.Table1FilterRecord(DataSet: TDataSet; var Ac-
cept: Boolean);
  begin
  Accept := DataSet['UchStepen'] = 'дтн';
  end;
```

то установка *Table1.Filtered в* True приведет к двум фильтрациям; в результирующем наборе данных будут показаны только записи, у которых поле Doljnost содержит значение 'профессор' и поле UchSicpen содержит значение 'дтн'.

Установка *Filtered* в False приведёт к отмене фильтрации, условия которой указаны в событии OnFilterRecord *или (и) в свойстве Filter.* При этом фильтрация, *наложенная на НД методом* SetRange *или* ApplyRange *и ему сопутствующими методами, не нарушается.*

Внимание! Сложность задания фильтра повышается при использовании ключа строкового типа. При этом следует помнить, что для таких случаев используется ***правило 2-х кавычек:***

3 кавычки подряд читаются как две кавычки (если поставить только 2 кавычки подряд, то получим пустую строку).

Пример формирования строки запроса для ввода условия запроса в Edit1:

```
Table1.Filter:='[Doljnost]='''+Edit1.Text+'''';
```

Пример. *Пусть* ТБД "Сотрудники" подвергается фильтрации

```
//DELPHI
procedure  Tform1.Table1FilterRecord(DataSet: TDataSet; var Ac-
cept: Boolean);
begin
Accept := DataSet['UchStepen'] = 'доцент';
end;
//C++
void __fastcall TForm1::Table1FilterRecord(TDataSet *DataSet,
bool &Accept)
{AnsiString s;
s="доцент";
Accept= (DataSet->FieldValues["UchStepen"] == s);
}
```

и по нажатию кнопки "SetRange" показываются только записи, у которых табельный номер больше 150000:

```
//DELPHI
procedure Tform1.SetRangeClick(Sender: TObject);
begin
Table1.SetRange([150000],[900000]);
end;
/C++
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Table1->SetRange
(ARRAYOFCONST((150000)),ARRAYOFCONST((900000)));
}
```

Нажатие клавиши "CancelRange" снимает фильтрацию по табельному номеру:

```
//DELPHI
procedure Tform1.CancelRangeClick(Sender: TObject);
begin
Table1.CancelRange;
end;
```

Последовательность установки фильтров произвольна - SetRange может применяться после Filtered:=True, и наоборот.

Отмена одного из этих условий фильтрации не приводит к отмене другого способа фильтрации:

Событие OnFilterRecord
property OnFilterRecord:TFilterRecordEvent;

Событие OnFilterRecord возникает, когда свойство Filtered устанавливается в True. Обработчик события *OnFilterRecord* имеет два параметра: имя фильтруемого набора данных и *var Accept*, указывающий условия фильтрации записей в НД. В отфильтрованный НД включаются только те записи, для которых параметр *Accept* имеет значение *True*.

В условие фильтрации могут входить любые поля НД, в том числе не входящие в текущий индекс, а также не входящие ни в один индекс. Возможность фильтрации НД по неиндексным полям, а также полям, не входящим в текущий индекс, выгодно отличает способ фильтрации с использованием события *OnFilterRecord* и свойства *Filtered от* способов фильтрации с использованием методов *SetRange, ApplyRange* и им сопутствующих методов (компонент TTable). Последние, как будет показано в разделе, посвященном компоненту TTable, позволяют производить фильтрацию НД только по индексным полям, входящим к тому же в состав индекса, текущего на момент фильтрации. Кроме этого, второй способ часто не позволяет реализовывать сложные логические конструкции при указании условий фильтрации.

Однако следует помнить о том, что при указании условий фильтрации НД в обработчике *OnFilterRecord*, в нем последовательно перебираются все записи ТБД при анализе их на предмет соответствия условию фильтрации, в то время как методы *SetRange, ApplyRange* и им сопутствующие методы используют индексно-последовательный метод доступа, т.е. работают с частью записей в физической ТБД. Это делает использование *OnFilterRecord* предпочтительным для небольших объемов записей и сильно ограничивает применение данного способа фильтрации при больших объемах данных.

Всякий раз, когда приложение обрабатывает событие *OnFilterRecord*, НД переводится из состояния *dsBrowse* в состояние *dsFilter*. Это предотвращает модификацию НД во время фильтрации. После завершения текущего вызова обработчика события *OnFilterRecord*, НД переводится в состояние *dsBrowse*.

Пример. Отфильтровать ТБД "Сотрудники" согласно условию "Показать всех доцентов":

```
procedure Tform1.Table1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
Accept := DataSet['Doljnost'] = 'доцент';
end;
//C++
void __fastcall TForm1::Table1FilterRecord(TDataSet *DataSet,
bool &Accept)
{AnsiString s;
s="доцент";
Accept= (DataSet->FieldValues["UchStepen"] == s);
}
```

Пример. Отфильтровать ТБД "Сотрудники" по условию "Показать всех сотрудников с табельным номером, вводимым в Edit1, и с вхождением в ФИО [символов, вводимых пользователем в Edit2":

```
//DELPHI
procedure Tform1.Table1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
Accept := (DataSet['TabNum'] > Edit1.Text))
AND (Pos(Edit2.Text,DataSet['FIO']) > 0);
end;
```

```
//C++
void __fastcall TForm1::Table1FilterRecord(TDataSet *DataSet,
bool &Accept)
{AnsiString s1,s2,s;
s1=Edit1->Text;
s2=Edit2->Text;
s= DataSet->FieldValues["TabNum"] ;
Accept= ((DataSet->FieldValues["FIO"] > s1)&&(s.Pos(s2)>0));
}
```

Методы *FindFirst, FindLast, FindNext, FindPrior* также используют свойство OnFilterRecord, когда выполняют навигацию по НД.

Свойство Filter

property Filter: string;

Свойство Filter позволяет указать условия фильтрации. В этом случае НД будет отфильтрован, как только его свойство Filtered станет равным True. Синтаксис похож на синтаксис предложения WHERE SQL-оператора SELECT с тем исключением, что: имена переменных программы указывать нельзя, можно указывать имена полей и литералы (явно заданные значения).

Можно применять операторы отношения:

< Меньше чем !
> Больше чем
>= Больше или равно
<= Меньше или равно
= Равно
<> Не равно

а также использовать логические операторы AND, NOT и OR:

([Doljnost] = 'доцент') AND ([TabNum] > 300000)

Строку фильтрации можно ввести во время выполнения:

когда проставляется галка в поле компонента CheckBox1 (то есть когда CheckBox1.Checked=True), пользователь выключает фильтрацию; когда пользователь снимает отметку (то есть когда CheckBox1.Checked=False),

```
//DELPHI
procedure Tform1.CheckBox1Click(Sender: TObject);
begin
Table1.Filter := Edit1.Text;
Table1.Filtered := CheckBox1.Checked;
end;
```

Однако при этом нужно следить, чтобы введенная строка соответствовала требованиям, предъявляемым к синтаксису строки *Filter.*

***Другим способом мог бы быть обработчик, считывающий значения фильтрации и преобразующий их к формату строки* Filter.**

Свойство FilterOptions

property FilterOptions:TFilterOptions;

TFilterOption = (foCaseInsensitive, foNoPartialCompare);

Свойство *FilterOptions* позволяет установить режимы фильтрации с использованием свойства *Filter.* По умолчанию FilterOptions = [];

foCaseInsensitive - Фильтрация производится без учета разницы в высоте букв;

foNoPartialCompare - поиск производится на точное соответствие. В противном случае, при фильтре FIO = 'Ма' в отфильтрованный НД будут включены записи, у которых в поле FIO частично входит 'Ма'. Например (если не используется *опция foCaseInsensitive*), 'Мануйлова' и 'Комарова'.

Навигация в неотфильтрованном НД между записями, удовлетворяющими фильтру

Методы *FindFirst, FindLast, FindNext, FindPrior* позволяют перемещаться в неотфильтрованном НД (у которого *Filtered* = False) между записями, удовлетворяющими условию фильтрации. Условие фильтрации задается событием *OnFilterRecord* или (и) свойством *Filter*. Действие данных методов таково: они кратковременно переводят НД в отфильтрованное состояние *(Filtered* = True) без визуализации этой фильтрации в TDBGrid или другом подобном компоненте, находят соответствующую запись и переводят НД в неотфильтрованное состояние *(Filtered* - False).

Если искомая запись найдена, данные методы возвращают True, в противном случае - False. Аналогичный результат возвращает свойство *Found*.

Т.е., для НД, в котором определены условия фильтрации, но сама фильтрация в текущий момент не включена, Delphi предоставляет интересную возможность. Она заключается в том, что в неотфильтрованном в данный момент НД можно обеспечить навигацию только между теми записями, которые удовлетворяют условию фильтрации (оно в текущий момент, когда свойство *Filtered*=False, не действует).

Для этой цели используются методы *FindFirst, FindLast, FindNext, FindPrior*.

function FindFirst: Boolean; - переходит на первую запись, удовлетворяющую фильтру;

function FindLast: Boolean; - переходит на последнюю запись, удовлетворяющую фильтру;

function FindNext: Boolean; - переходит на следующую запись, удовлетворяющую фильтру;

function FindPrior: Boolean; - переходит на предыдущую запись, удовлетворяющую фильтру;

property Found: Boolean; - возвращает True, если последнее обращение к одному из методов FindFirst, FindLast, FindNext, FindPrior привело к нахождению нужной записи.

Пример. Предоставить пользователю возможность перемещаться на первую, последнюю, следующую, предыдущую запись, удовлетворяющую условию "Содержимое Edit1 входит как часть ФИО сотрудника". Заметим, что свойство *Table1.Filtered = False*, т.е. хотя в обработчике события Table1.OnFilterRecord и указано условие фильтрации, в НД показываются все записи, и он остается в неотфильтрованном состоянии.

```
//условие фильтрации
//DELPHI
procedure TFormX.Table1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
Accept := POS(Edit1.Text,DataSet['FIO']) > 0;
end;
// нажата кнопка "Первая"
procedure TFormX.FindFirstButtonClick(Sender: TObject);
begin
Label1.Caption :='';
IF not Table1.FindFirst THEN
Label1.Caption :='Нет такой записи';
```

```
end;
//C++
void __fastcall TForm1::Button2Click(TObject *Sender)
{
if (!Table1->FindFirst()) Label1->Caption="Нет такой записи";
}

// нажата кнопка "Последняя"
procedure TFormX.FindLastButtonClick(Sender: TObject);
begin
Label1.Caption :='';
IF not Table1.FindLast THEN
Label1.Caption := 'Нет такой записи';
end;
// нажата кнопка "Следующая"
procedure TFormX.FindNextButtonClick(Sender: TObject);
begin
Label1.Caption :='';
IF not Table1.FindNext THEN
Label1.Caption := 'Нет такой записи';
end;
// нажата кнопка "Предыдущая"
procedure TFormX.FindPriorButtonClick(Sender: TObject);
begin
Label1.Caption :='';
IF not Table1.FindPrior THEN
Label1.Caption :='Нет такой записи';
end;
```

Заметим, что поскольку фильтрация записей с использованием события *OnFilterRecord или* (и) свойства *Filter* может применяться только на небольших объемах записей (из-за того, что при этом используется последовательный метод доступа к записям в ТБД), аналогичные ограничения накладываются и на поиск записей с использованием методов *FindFirst, FindLast, FindNext, FindPrior.*

Задание

Таблицы для всех лабораторных работ выбираются в соответствии с вариантом задания.

Создать форму для просмотра данных таблицы. Дополнить форму элементами управления (создать интерфейс) для реализации поиска и отбора данных.

ПОИСК ДАННЫХ

1) Определить условие поиска по любому неключевому полю (например, найти сотрудника указанной должности).

Организовать поиск записи по ключевому значению с помощью метода Locate компонента TTable. В чем заключается результат поиска?

2) Создать индексный файл по выбранному в п.1 полю с помощью Dekstop - Restructure - Secondary Index

Добавить на форму 2 кнопки, обработчиком события OnClick которых является установка или отмена текущего индекса.

3) Осуществить поиск записей по такому же условию:

в проиндексированном файле с помощью метода FindKey;

неточный поиск с помощью метода FindNearest;

Дополнить форму полем ввода данных для поиска (должности сотрудника);

Организовать инкрементальный локатор с помощью метода FindNearest для поиска записи.

Локатор – механизм поиска с позиционированием курсора. Инкрементальный локатор по вводу каждого символа переходит на запись, ближайшую к искомой.

Изучить результат позиционирования текущей записи при вводе ключевого значения поиска, которого нет в таблице, для всех 3-х вариантов поиска (что будет, если искать отсутствующего автора Иванов?)

Чем отличаются методы поиска FIND... от метода Locate?

4) Осуществить поиск записей по такому же условию, не перемещая текущий указатель. Использовать метод LookUp. Присвоить переменной L значение какого-либо поля найденной записи (например, фамилии сотрудника).

ОТБОР ДАННЫХ

5) Сформулировать условие отбора данных, например, отобрать всех сотрудников с должностью Профессор.

Установить фильтр двумя способами:

- свойство Filter компонента TTable (при Filtered = True)
- событие OnFilterRecord

Организовать интерфейс для ввода условия фильтрации с клавиатуры.

Предоставить пользователю возможность перемещаться между записями, удовлетворяющими фильтру, при отключенной фильтрации. (Методы FindFirst, FindLast, FindNext, FindPrior).

6) Отфильтровать записи в проиндексированном файле, используя методы SetRange, CancelRange.

Содержание отчета

Цель работы

Содержимое записей таблицы

Для каждого задания постановка задачи поиска или отбора (логическое выражение), использованный метод в конкретном применении (команда программы), результат поиска или отбора.

Контрольные вопросы

1. В чем заключается результат поиска?
2. Методы поиска в проиндексированных и неупорядоченных наборах данных.
3. Какие данные отображаются в результате фильтрации?
4. Как связаны свойства Filter и Filtered набора данных?
5. Каков результат поиска записи, которой нет в таблице?
6. Чем характеризуется неточный поиск для строкового и числового значения?
7. Когда наступает событие OnFilterRecord?

Лабораторная работа № 4

Использование SQL- запросов В среде Delphi

Цель работы. Получить навыки практического использования языка SQL

Методические рекомендации

Для формирования набора данных, содержащего выборки из одной или нескольких таблиц, используется компонент TQuery. Основное свойство компонента – SQL- хранит текст запроса на языке SQL, который и является определяющим для результата запроса.

Для организации соединения компонента с БД необходим источник данных. Для просмотра результата необходим визуальный компонент (например, TDBGrid).

Используются следующие свойства компонент (аналогично просмотру данных компонента TTable):

TdataSource.DataSet

TDBGrid.DataSource

Tquery.DataBaseName, Tquery.Active

Тексты программ приведены для таблицы, хранящейся в файле с именем Country.db из БД DBDEMOS (стандартный пример).

	Name	Capital	Continent	Area	Population
	Argentina	Buenos Aires	South America	2 777 815,00	32 300 003,00
	Bolivia	La Paz	South America	1 098 575,00	7 300 000,00
	Brazil	Brasilia	South America	8 511 196,00	150 400 000,00
	Canada	Ottawa	North America	9 976 147,00	26 500 000,00
	Chile	Santiago	South America	756 943,00	13 200 000,00
	Colombia	Bagota	South America	1 138 907,00	33 000 000,00
	Cuba	Havana	North America	114 524,00	10 600 000,00

Формирование запроса

--ручной способ;

набрать текст запроса на языке SQL в окне текстового редактора свойства SQL компонента Tquery.

--с помощью Visual Query Builder;

в контекстном меню компоненты TQuery выбрать пункт SQL Builder...(построитель запросов).

Здесь же можно просмотреть результат выполнения запроса и сформированный текст на SQL, используя панель инструментов.

Статические и динамические запросы

Если свойство Active установлено в окне инспектора объектов, то результат выполнения запроса изменить нельзя в течение времени работы приложения. Это запрос статический. Для изменения результатов запроса следует использовать какое-либо событие для активизации запроса. Такой запрос является динамическим, т.к. может быть изменен в течение времени выполнения приложения.

Параметры динамического запроса

Для ввода условий отбора с клавиатуры используют параметры динамического запроса (свойство Params компонента Tquery). Запрос с параметрами нельзя создать с помощью построителя запросов, т.к. SQL Builder не поддерживает это свойство.

Для использования параметра отбора следует создать текст SQL с параметром, например:

```
SELECT Name, Capital, Continent
FROM "country.db"
where (Name= :0)
```

Последняя строка может содержать не номер, а имя параметра: where (Name= :coun)

При наступлении определенного события следует значению параметра присвоить введенное значение и активизировать запрос:

```
//DELPHI
query2.Params[0].asstring:=edit1.text  ;
query2.active:=true; {или query2.open;}

//C++
Query1->Params->Items[0]->AsString=Edit1->Text;
Query1->Active=true;
```

Активизация (выполнение запроса) может происходить 2-мя способами:

--Установкой свойства Active в значение True;

--Выполнением метода Open.

Формируемые запросы

Свойство SQL может быть сформировано во время выполнения приложения. Для этого используются методы Add, Delete, Clear. Тогда один компонент Tquery может быть использован для выполнения различных отстоящих друг от друга по времени запросов. Это уменьшает количество используемых компонент, но увеличивает объем кода программы. Пример:

```
//DELPHI
   with query2 do begin
  SQL.clear;                    {очистить список строк запроса SQL}
        SQL.add ('select name');      {добавить текст строки запроса SQL}
        SQL.add ('from "Country.db"');
   open;                        {выполнить запрос SQL}
end;

//C++
Query1->SQL->Clear();
Query1->SQL->Add("Select Name");
Query1->SQL->Add("from Country.db");
Query1->Active=true;
```

Активные запросы

Активным называется запрос, который изменяет содержимое таблицы. Виды активных запросов:

Добавление записей; (метод INSERT)

Изменение записей; (метод UPDATE)

Удаление записей. (метод DELETE)

Создание таблицы (метод CREATE).

Внимание! Построитель запросов SQL Builder не поддерживает выполнение данных операторов!!

Примеры активных запросов:

Удаление из таблицы записи со значением поля Name="Russia"	`delete from "country.db" where (name="russia")`
Добавление записи с указанным значением полей в таблицу	`Insert into "country.db" (Name,Capital) Values ("Russia","Moscow")`
Увеличение занимаемой площади и численности населения на 1 для стран, названия которых начинаются на букву B	`Update "country.db" set area=area+1, popultion=population+1 where (name>="B") And (name<="C")`

Для выполнения активных запросов следует использовать метод ExecSQL, например, при нажатии кнопки:

```
//DELPHI
procedure TForm1.Button2Click(Sender: TObject);
begin
query1.ExecSQL;
end;
```

Внимание! В отличие от оператора Select активные запросы не возвращают набор данных! Результат выполнения запроса – действие над таблицей. Для просмотра результата действия в учебных целях или при отладке рекомендуется организовать просмотр данных таблицы стандартным способом и обновлять информацию таблицы после каждого выполнения запроса, например, по нажатию кнопки:

```
//DELPHI
procedure TForm1.Button1Click(Sender: TObject);
begin
table1.Refresh;
end;
```

Задание

Таблицы для всех лабораторных работ выбираются в соответствии с вариантом задания!

1. Спроектировать функции приложения своей предметной области, для реализации которых следует использовать запросы.
2. Реализовать запросы в среде Delphi.

Работа должна содержать 7 запросов следующего вида:

1. Запрос на выборку данных с несколькими условиями, реализованный через свойство SQL;
2. Запрос на выборку данных из двух объединенных таблиц, реализованный с помощью SQL Builder;
3. Динамический запрос с вводом условия отбора с клавиатуры;
4. Формируемый запрос: реализовать 1 и 2 запросы с использованием одного компонента Tquery;
5. Три активных запроса: на добавление, удаление и обновление данных в таблице.

Пример проекта для БД "DBDEMOS":

1. В таблице «Country.db» найти страны Северной Америки, население которых составляет 10-20 млн.
2. Для всех клиентов из таблицы «Clients.db» указать страну проживания и численность ее населения.
3. Континент и численность населения для 1 запроса ввести с клавиатуры.
4. Формировать запросы 1 и 2 по нажатию кнопок в одном компоненте Tquery.

См. таблицу примеров активных запросов.

Содержание отчета

1. Тема и цель работы
2. Постановка задачи и текст каждого запроса.

Контрольные вопросы

1. Чем отличаются статические и динамические запросы, активные и неактивные запросы?
2. Назначение параметров запроса.
3. Методы выполнения запросов.
4. Чем отличается метод Open от задания свойства Active при выполнении статического запроса?
5. Какое свойство объекта Tquery хранит текст SQL запроса?
6. Как просмотреть результат выполнения запроса?

Лабораторная работа № 5

Построение отчетов

Цель работы. Изучить общую структуру отчета БД. Получить навыки генерации отчетов.

Методические рекомендации

Компоненты для построения отчетов

В Delphi на странице палитры компонентов *QReport* расположено около двух десятков компонентов, применяемых для построения отчетов. "Главным" компонентом, несомненно, является ***TQuickRep,*** определяющий поведение отчета в целом. Другие компоненты определяют составные части отчета:

TQRBand - заготовка для расположения данных, заголовков, титула отчета и др.; отчет, в основном, строится из компонентов *TQRBand,* которые реализуют:

область заголовка отчета;
область заголовка страницы;
область заголовка группы;
область названия столбцов отчета;
область детальных данных, предназначенную для отображения данных самого нижнего уровня детализации;
область подвала группы;
область подвала страницы;
область подвала отчета.

TQRSubDetail - определяет область, в которой располагаются данные подчиненной таблицы при реализации в отчете связи Master-Detail на основе существующей связи между ТБД;

TQRGroup - применяется для группировок данных в отчете;

TQRLabel - позволяет разместить в отчете статический текст;

TQRDBText - позволяет разместить в отчете содержимое поля набора данных;

TQRExpr - применяется для вывода значений, являющихся результатом вычисления выражений; алгоритм вычисления выражений строится при помощи редактора формул данного компонента;

TQRSysDate - служит для вывода в отчете даты, времени, номера страницы, счетчика повторений какого-либо значения и т.д.;

TQRMemo - служит для вывода в отчете содержимого полей комментариев;

TQRRichText - служит для вывода в отчете содержимого полей форматированных комментариев;

TQRDBRichText - служит для вывода в отчете содержимого полей форматированных комментариев, источником которых является поле набора данных;

TQR.Shape - служит для вывода в отчете графических фигур, например, прямоугольников;

TQRImage - служит для вывода в отчете графической информации, источником которой является поле набора данных;

TQRChart - служит для встраивания в отчет графиков.

Компонент TQuickRep

Компонент *TQuickRep* определяет поведение и характеристики отчета в целом. При размещении этого компонента в форме в ней появляется сетка отчета. В дальнейшем в этой сетке располагаются составные части отчета, например, группы *TQRBand.*

Важнейшие свойства, методы и события компонента TQuickRep.

Свойства

property Bands: TQuickRepBands;

состоит из множества логических значений (False/True), которые определяют включение в отчет отдельных видов составляющих:

- *HasColumnHeader* - заголовка столбцов отчета;

- *HasDetail* - детальной информации;
- *HasPageFooter* - подвала страницы;
- *Has Page Header* - заголовка страницы;
- *HasSummary* - подвала отчета;
- *HasTitle* - заголовка отчета.

property DataSet: TDataSet;

указывает на набор данных, на основе которого и создается отчет.

Обычно для выдачи отчета используется один НД. Если нужно вывести связанную информацию из нескольких таблиц БД, ее объединяют в одном НД при помощи оператора SELECT. В этом случае в качестве НД для отчета может использоваться компонент TQuery. Информацию из нескольких связанных НД можно включать в отчет, если эти наборы данных связаны в приложении отношением Master-Detail. В этом случае в качестве НД отчета указывается Master-набор, а ссылка на соответствующие Detail-наборы осуществляется в компонентах TQRSubDetail. Если в отчет нужно включить информацию из несвязанных наборов данных, применяют композитный отчет, то есть отчет, составленный из группы других отчетов.

property Frame : TQRFrame;

определяет параметры рамки отчета:

- *Color* - цвет линии рамки;
- *DrawBottom* - определяет, следует ли выводить линию снизу;
- *DrawLeft*- определяет, следует ли выводить линию слева;
- *DrawRight*- определяет, следует ли выводить линию справа;
- *DrawTop*- определяет, следует ли выводить линию сверху;
- *Style* - определяет стиль линии;
- *Width* - определяет ширину линии в пикселях.

property Page: TQRPage;

определяет параметры страницы.

property PrinterSettings: TQuickRepPrinterSettings;

определяет параметры принтера.

property PrintIfEmpty: Boolean;

указывает (True), что следует печатать отчет даже в том случае, если он не содержит данных.

Методы

procedure NewPage;

Выполняет переход на новую страницу. Может использоваться в обработчиках событий компонентов отчета *BeforePrint* или *AfterPrint* и не может - в обработчиках событий *OnPrint*, *OnStartPage* и *OnEndPage*.

procedure Preview;

выводит отчет в окно предварительного просмотра.

Чтобы во время разработки отчета просмотреть в окне предварительного просмотра содержимое отчета в том виде, как он будет выводиться на печать, необходимо:

- выбрать отчет при помощи мыши;
- нажать правую кнопку мыши;
- во всплывающем меню выбрать элемент *Preview*.

Следует заметить, что при этом не будут видны некоторые данные, например, значения вычисляемых полей наборов данных. Они будут выводиться только во время выполнения.

procedure Print;

печатает отчет на принтере.

События

property AfterPreview : TQRAfterPreviewEvent;

наступает после закрытия окна предварительного просмотра отчета.

property AfterPrint: TQRAfterPrintEvent;
наступает после вывода отчета на печать.
property BeforePrint: TQRBeforePrintEvent;
наступает в момент генерации отчета, до выдачи окна предварительного просмотра отчета и до вывода отчета на печать.
property OnEndPage : procedure(Sender : TObject);
наступает в момент подготовки к генерации последней страницы отчета.
property OnStartPage : procedure(Sender : TObject);
наступает в момент подготовки к генерации первой страницы отчета.

Компонент TQRBand
Компоненты *TQRBand* являются основными составными частями отчета и используются для размещения в них статического текста и данных. Месторасположение компонента в отчете и его поведение определяются свойствами

property BandType: TQRBandType;
Ниже перечислены возможные значения этого свойства.
• *rbTitle* - определяет компонент заголовка отчета. Информация, размещенная в компоненте TQRBand, располагается перед всеми другими частями отчета. Этот вид компонента TQRBand используется для вывода заголовочной информации отчета.
• *rb Page Header* - определяет компонент заголовка страницы. Информация, размещенная в компоненте с этим значением свойства BandType, выводится всякий раз при печати новой страницы отчета прежде всех иных частей отчета (но после информации, размещенной в компоненте заголовка отчета - для первой страницы).
• *rbDetail* - компонент детальной информации. Выводится всякий раз при переходе на новую запись в НД отчета. Отчет печатается для всех записей НД, определяемого свойством отчета *DataSet*, начиная с первой записи и заканчивая последней. Позиционирование на первую запись и последовательный перебор записей в НД осуществляется компонентом TQuickRep автоматически.
• *rbPageFooter* - компонент подвала страницы. Выводится для каждой страницы отчета после всех иных данных на странице.
• *rbSummary* - компонент подвала отчета. Выводится на последней странице отчета после всей иной информации, но перед подвалом последней страницы отчета.
• *rbGroupHeader* - компонент заголовка группы. Применяется при группировках информации в отчете. Выводится всякий раз при выводе новой группы.
• *rbGroupFooter* - компонент подвала группы. Применяется при группировках информации в отчете. Выводится всякий раз при окончании вывода группы, после всех данных группы.
• *rbSubDetail* - компонент для выдачи детальной информации из подчиненного набора данных, при выводе в отчете информации из двух или более наборов данных, связанных в приложении при помощи механизма Master-Detail. Это значение присваивается компоненту автоматически, когда генерируется компонент TQRBand при размещении в форме компонента TQRSubDetail. Программа не должна устанавливать это значение в свойство *BandType*.
• *rbColumnHeader* - компонент для размещения заголовков столбцов. Размещается в отчете на каждой странице после заголовка страницы.
• *rbOverlay* - используется для совместимости с более ранними версиями отчетов.

property Enabled : Boolean;
указывает, печатается в отчете (True) или нет (False) информация, содержащаяся в компоненте TQRBand.

Событие
property BeforePrint: TQRBeforePrintEvent;
наступает перед печатью информации, размещенной в области компонента TQRBand.

Создание простейшего отчета

Компоненты TQuickRep и TQRBand являются минимально достаточными для создания простого отчета, не содержащего внутри себя группировок информации.

Для создания отчета нужно использовать компонент TQuickRep и компоненты TQRBand для каждой области отчета (заголовка отчета, заголовка страницы и т.д.). Для каждого компонента TQRBand следует добавить компонент отображения данных, например, Tlabel или TDBText., TQRExpr для вычисления выражений.

Использование TQRBand для представления заголовков столбцов

Компонент TQRBand, у которого в свойство BandType установлено значение rbColumnHeader, используется для представления заголовков столбцов. Заголовки столбцов определяются при помощи компонентов TQRLabel.

Использование TQRBand для показа заголовка и подвала страницы

Компонент TQRBand, у которого в свойство BandType установлено значение rbPageHeader, используется для показа заголовка страницы. Он выводится для каждой новой страницы перед выводом другой информации. Компонент TQRBand, у которого в свойство BandType установлено значение rbPageFooter, используется для показа подвала страницы. Он выводится для каждой страницы после вывода любой иной информации.

Информация в заголовке и подвале страницы может формироваться на основе статического текста (компоненты TQRLabel), значений полей (компоненты TQRDBText) и результатов вычисления выражений (компоненты TQRExpr).

Использование компонента TQRSysData для показа вспомогательной и системной информации

Компонент TQRSysData используется для показа вспомогательной и системной информации. Вид показываемой информации определяется свойством

property Data : TQRSysDataType;

Ниже указаны возможные значения этого свойства.

- *qrsColumnNo* - номер текущей колонки отчета (для одноколоночного отчета всегда 1).
- *qrsDate* - текущая дата.
- *qrsDate Time* - текущие дата и время.
- *qrsDetailCount* - число записей в НД; при использовании нескольких НД - число записей в master-наборе. Для случая, когда НД представлен компонентом TQuery, эта возможность может быть недоступной, что связано с характером работы компонента TQuery, который возвращает столько записей, сколько необходимо для использования в текущий момент, а остальные предоставляет по мере надобности;
- *qrsDetailNo* - номер текущей записи в НД. При наличии нескольких наборов - номер текущей записи в master-наборе.
- *qrsPageNumber* - номер текущей страницы отчета.
- *qrsPageCount* - общее число страниц отчета.
- *qrsReportTitle* - заголовок отчета.
- *qrsTime* - текущее время.

Группировки данных в отчете

Группировка- это изменение порядка расположения записей с учетом условия группировки. Например, если добавить группировку по полю Континент, то страны, расположенные на одном континенте, попадут в одну группу и будут печататься рядом. Т.е. группировка объединяет записи с одинаковым значением поля. Кроме визуально удобного расположения, это дает возможность вычислять итоги по группе (например, количество стран на каждом континенте). Итоги размещаются в подвале группы.

Для группировки информации используется компонент TQRGroup. Его свойство Expression указывает выражение. В группу входят записи НД, удовлетворяющие условию выражения. При смене значения выражения происходит смена группы. Для каждой

группы, если определены, выводятся заголовок группы и подвал группы. В качестве заголовка группы служит компонент TQRBand со значением свойства BrandType, равным *rbColumnHeader*. В качестве подвала группы служит компонент TQRBand со значением свойства BrandType, равным rbGroupFooter.

Свойство FooterBand компонента TQRGroup содержит ссылку на компонент подвала группы.

В заголовке группы, как правило, выводится выражение, по которому происходит группировка, и различные заголовки, если они нужны. В подвале группы обычно выводится агрегированная информация – суммарные, средние и т.п. значения по группе.

Построение отчета на основе нескольких наборов данных, связанных в приложении как Master – Detail

Если необходимо выдавать отчет на основе более чем одной ТБД, можно поступить двумя способами:

1. В рамках компонента TQuery произвести соединение данных из нескольких таблиц БД в один НД, после чего определить в отчете нужные группировки;

2. Создать в приложении по одному НД на каждую таблицу БД, соединить эти наборы между собой связью Master-Detail (используя свойства *MasterSource, MasterFields* набора данных) и применить в отчете компонент (или несколько компонентов) TQRSubDetail для вывода информации из подчиненного (Detail) НД (или группы подчиненных НД), для вывода информации из основного (Master) НД, как и в обычных отчетах, применяется компонент TQRBand, у которого в свойство *BandType* установлено значение *rbDetail*.

Компонент TQRSubDetail предназначен для показа информации в отчете из подчиненного НД. Его свойство

property DataSet: TDataSet;

указывает имя подчиненного НД, информация из которого будет выводиться в пространстве компонента TQRSubDetail. В остальном использование данного компонента аналогично использованию компонента TQRBand, у которого значение в свойство *BandType* установлено значение *rbDetail*.

Пример. Пусть имеется таблица БД Tovary.DB, содержащая помимо прочих поле Tovar (название товара).

Пусть также имеется таблица БД Rashod.DB, содержащая сведения об отпуске материалов со склада. В состав ТБД входят поля

- N_RASH - уникальный номер события отпуска товара;
- DEN -номер дня;
- MES - номер месяца;
- GOD - номер года;
- TOVAR - наименование отпущенного товара;
- POKUP - наименование покупателя;
- K.OLVO - количество единиц отпущенного товара.

Таблицы Tovary.DB и Rashod.DB находятся в отношении "один-ко-многим"; то есть одному товару может соответствовать более одного факта отпуска товара со склада.

Разместим в форме компонент TTable (имя TovaryTable), ассоциированный с ТБД Tovary.DB. Разместим в форме компонент TDataSource (имя DS_TovaryTable) и свяжем его с TovaryTable. Разместим в форме компонент TTable (имя RashodTable), ассоциированный с ТБД Rashod.DB.

Установим связь Master-Detail в приложении между НД TovaryTable и RashodTable. Для этого при помощи редактора связей установим в свойство *RashodTable. MasterSource* значение DS_TovaryTable, и в свойство *RashodTable. MasterFields* значение 'TOVAR'.

Задание

Для выполнения всех лабораторных работ таблицы выбираются в соответствии с вариантом задания.

1) Для выбранной таблицы составить отчет, включающий следующую информацию:

- Заголовок отчета;
- № страницы отчета;
- заголовки столбцов на каждой странице;
- записи таблицы;
- количество записей на каждой странице;
- среднее значение любого числового поля в конце отчета.

2) Выполнить группировку данных в отчете по выбранному полю. В заголовок группы включить произвольный текст и значение поля группировки, из записей поле группировки исключить.

Проверить визуально правильность формирования групп, учесть необходимое предварительное условие упорядочения данных по полю группировки. Вычислить количество записей в каждой группе и разместить это значение в подвале группы.

3) Выбрать вторую таблицу, в которой содержится уточняющая информация для поля первой таблицы. Построить отчет на основе данных двух таблиц двумя способами:

создать один объединяющий набор данных TQuery и использовать соответствующие группировки в отчете;

связать вторую таблицу с первой соотношением Master-Detail и построить отчет с использованием уточняющей информации в виде подчиненного набора данных.

ПРИМЕР.

Таблица1

	Страна	Столица	Континент
1	Argentina	Buenos Aires	South America
2	Bolivia	La Paz	South America
3	Brazil	Brasilia	South America
4	Canada	Ottawa	North America
5	Chile	Santiago	South America
6	Colombia	Bagota	South America
7	Cuba	Havana	North America

Таблица 2

	Страна	Область	Население
1	Argentina	X	123
2	Argentina	Y	345
3	Cuba	X	23
4	Cuba	Y	234

Отчет должен содержать информацию о населении областей каждой страны.

Содержание отчета

1. Тема и цель работы.
2. Рисунок общей структуры отчета.
3. Постановка задачи, структура, использованные компоненты, условия группировки для каждого отчета.

Контрольные вопросы

1. Структура отчета (8 областей).
2. Компоненты Delphi, используемые для организации отчета, и их назначение.
3. Что такое группировка данных в отчете? Какие данных объединяются в одну группу?
4. Для чего используют группировку?
5. Способы построения отчета с детализирующей информацией.

Варианты заданий для лабораторных работ по курсу "Базы и банки данных"

Для каждого варианта представлена логическая модель БД, состоящей из 3-х таблиц. Структура каждой таблицы записана в строку. Ключи выделены жирным шрифтом.

Вариант 1 - Автомобили

1	**код модели**	марка	модель	Производитель	
2	**код модели**	**№кузова**	цвет	Пробег	цена
3	**№кузова**	год выпуска	объем		

Вариант 2 - Видеотека

1	**№клиента**	ФИО	телефон	Адрес
2	**№клиента**	**№фильма**	дата	
3	**№фильма**	режиссер	название	год выпуска

Вариант 3 - Автолюбители

1	**№авто**	марка	год произв.	Цена
2	**№паспорта**	ФИО	адрес	Телефон
3	**№авто**	**№паспорта**	дата постановки на учет	

Вариант 4 - Спортивные игры

1	**Команда**	**Страна**	Капитан	Тренер	
2	**№стадиона**	Город	страна	название стад	вместимость
3	**№стадиона**	команда	дата игры	время игры	

Вариант 5 - Услуги

1	**Код услуги**	название	стоимость	Продолжительность
2	**Фирма**	адрес	телефон	
3	**Код услуги**	**фирма**	**дата заказа**	кол.заказов

Вариант 6 - Студенты

1	**№зачетки**	ФИО	группа	
2	**Группа**	факультет	специальность	
3	**ФИО**	год рождения	адрес	Телефон

Вариант 7 - Итоги успеваемости студентов

1	**Код спец**	факультет	специальность	
2	**Группа**	код спец		
3	**Группа**	**дисциплина**	дата экзамена	средний балл

Вариант 8 - Успеваемость студентов

1	**№зачетки**	**№экз**	оценка		
2	**№зачетки**	**группа**	ФИО		
3	**№экз**	Группа	дисциплина	дата экзамена	средний балл

Вариант 9 - Отгрузка товаров

1	**№товара**	наименование	ед.измерения	Цена
2	**№накладной**	**№товара**	количество	Стоимость
3	**№накладной**	отв.лицо	дата отгрузки	средний балл

Вариант 10 - Продажа товаров

1	**№товара**	наименование	ед.измерения	Цена
2	**№чека**	**№товара**	количество	Стоимость
3	**№чека**	дата покупки	№кассы	Продавец

Вариант 11 - Расписание уроков

1	**Класс**	**Предмет**	часов в неделю	
2	Класс	Буква	**Предмет**	№учителя
3	№учителя	ФИО	специальность	Стаж

Вариант 12 - Детский сад

1	**№сада**	Адрес	ФИО завед.	
2	Категория	Мин.возраст детей	Макс.возраст детей	
3	**№сада**	**№группы**	Категория	Воспитатель

Вариант 13 - Игры в детском саду

1	**№игрушки**	Наименование	цвет	Категория	год покупки
2	**Категория**	**Мин.возраст детей**	Макс.возраст детей		
3	**№группы**	**№игрушки**			

Вариант 14 - Зоопарк

1	**№жив**	кличка	окрас	необх.площадь в клетке	необх. надежность
2	**№клетки**	**№жив**			
3	**№ клетки**	площадь	надежность		

Вариант 15 - Склад товаров

1	**№склада**	**№товара**	количество		
2	**№ товара**	наименование	ед.измерения	производитель	
3	**Производитель**	Адрес	телефон	Страна	

Вариант 16 - Библиотека

1	**шифр книги**	автор	название	Издательство	год
2	**№читателя**	ФИО	адрес	Телефон	образование
3	**№читателя**	**шифр книги**	дата выдачи	дата возврата	

Вариант 17 - Библиотека

1	**шифр книги**	автор	название	Издательство	год
2	**№библиотеки**	адрес	телефон		
3	**№библиотеки**	**шифр книги**	наименование зала		

Вариант 18 - Мобильная связь

1	**производитель**	**модель**	разрешение дисплея	Полифония	камера
2	**№дилера**	название фирмы	отв.лицо	Телефон	
3	**№дилера**	**производитель**	**модель**	Цена	срок гарант

Вариант 19 - Мобильная связь

1	**№офиса**	адрес	телефон	поставщик услуг
2	**№клиента**	ФИО	паспорт	год рождения
3	**№клиента**	**№офиса**	дата подкл	тарифный план

Вариант 20 - Мобильная связь

1	**Тарифный план**	абон.плата	стоимость1сек	стоимость sms
2	**№клиента**	ФИО	паспорт	год рождения
3	**№клиента**	**Тарифный план**	дата подкл	место подключения

Вариант 21 - Видеотека

1	**№кассеты**	название	жанр	Режиссер	год выпуска
2	**№паспорта**	ФИО	адрес	Телефон	
3	**№заказа**	**№кассеты**	дата выдачи	дата возврата	**№паспорта**

Вариант 22 Спортивная команда

1	**№игрока**	ФИО	Дата рождения	
2	**№игрока**	**команда**	амплуа	
3	**команда**	тренер	город	Спонсор

Вариант 23 Срок годности товаров

1	**Тип срока год**	Срок годности товаров		
2	**код товара**	**дата поставки**	количество	
3	**код товара**	тип срока год	цена	Наименование тов

Вариант 24 Клуб любителей животных

1	**№жив**	кличка	порода	Возраст	окрас	пол
2	порода	класс	тип			
3	**порода**	**возраст**	кол. Еды			

Вариант 25 - Мобильная связь

1	**№плана**	название	ст.1.мин.	ст. SMS
2	**№паспорта**	ФИО	адрес	
3	**№подключения**	№паспорта	№плана	дата подключения

Вариант 26 - Заказ товаров

1	**Код товара**	название	цена	Вес	
2	**№машины**	**водитель**	**марка**		
3	**код заказа**	код товара	№машины	Дата	кол

Вариант 27 - Заказ товаров

1	**№водителя**	ФИО	класс	Телефон
2	**№машины**	№водитель	марка маш	Модель
3	**марка маш**	**модель**	грузоподъемность	

ЛИТЕРАТУРА

1. Фленов М. Библия для программиста в среде Delphi. – СПб.: Питер, 2003.
2. Шумаков В.П. Delphi 3 и создание приложений баз данных.- Спб.: BHV, 1999.

УЧЕБНОЕ ИЗДАНИЕ

Составитель: Горбашко Лариса Ашотовна

Средства обработки баз данных в средах программирования Delphi и C++Builder

Методические указания к выполнению лабораторных работ
по дисциплине **"Базы и банки данных"**
для студентов специальности
1 530102 «Автоматизированные системы обработки информации»

Ответственный за выпуск: Горбашко Л.А.
Редактор: Строкач Т.В.
Компьютерная верстка: Боровикова Е.А.
Корректор: Никитчик Е.В.

Подписано к печати 29.12.2006 г. Бумага «Снегурочка». Формат 60x84 1/16. Заказ № 37.
Усл. п. л. 2,1. Уч.-изд. л. 2,25. Тираж 100 экз. Отпечатано на ризографе учреждения образования «Брестский государственный технический университет».
224017, г. Брест, ул.Московская, 267.