

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Нижегородский государственный университет
им. Н.И. Лобачевского**

В.Л. Тарасов

Работа с базами данных в Access 2010

Часть 2

Учебно-методическое пособие

Рекомендовано методической комиссией механико-математического
факультета для студентов ННГУ, обучающихся по направлениям подготовки
010200 "Математика и компьютерные науки",
010400 "Прикладная математика и информатика",
010800 «Механика и математическое моделирование»

**Нижний Новгород
2014**

УДК УДК 681.3.06
ББК 32.973.233
Т19

Т19 Тарасов В.Л. РАБОТА С БАЗАМИ ДАННЫХ В ACCESS 2010. ЧАСТЬ 2: Учебно-методическое пособие. – [электронный ресурс]. – Нижний Новгород: Нижегородский госуниверситет, 2014. – 126 с.

Рецензент: заведующий кафедрой информационных систем и технологий ННГАСУ
д.ф.-м.н., профессор **А.Н. Супрун**

В пособие изложены основные понятия, используемые при работе с базами данных и работа с базами данных в СУБД Access версии 2010. Описаны разработка таблиц, создание запросов на выборку и изменение данных, перекрестных запросов. Рассмотрено создание форм, отчетов, импорт данных из Access в другие приложения, создание документов слияния, макросов. Учебно-методическое пособие предназначено для студентов, изучающих базы данных

Ответственный за выпуск:
председатель методической комиссии
механико-математического факультета ННГУ,
к.ф.м.н., доцент **Н.А. Денисова**

УДК УДК 681.3.06
ББК 32.973.233

© Тарасов В.Л., 2014
© Нижегородский государственный
университет им. Н.И. Лобачевского, 2014

Содержание

Содержание.....	3
Предисловие	5
1. Управление приложением Access.....	6
1.1. Главная кнопочная форма	6
1.2. Анализ базы данных.....	10
1.2.1. Анализ таблиц	10
1.3. Защита базы данных.....	15
1.3.1. Установка пароля	15
1.3.2. Удаление пароля	16
2. Модули	18
2.1. Первая программа	18
2.2. Краткий обзор языка VBA.....	20
2.2.1. Алфавит и лексика	20
2.2.2. Ключевые слова	20
2.2.3. Операции	20
2.2.4. Типы данных	21
2.2.5. Переменные, константы, массивы	21
2.2.6. Операторы цикла и условия	21
2.2.7. Процедуры и функции	22
2.3. Среда разработки	23
2.3.1. Состав среды	23
2.4. Отладка.....	24
2.5. Управление структурой проекта	25
2.6. Справка.....	25
2.7. Классы и объекты	25
2.7.1. Объявление типов	25
2.7.2. Понятие класса	26
2.7.3. Пример класса	26
2.7.4. Объекты и ссылки на объекты	28
2.7.5. Иерархия классов	28
2.7.6. Скрытие данных	29
2.8. Полиморфизм.....	30
2.8.1. Оператор With	31
2.9. Пример обработки данных на VBA.....	31
2.9.1. Подключение библиотек объектов	32
2.9.2. Постановка задачи.....	33
2.9.3. Алгоритм решения задачи.....	34
2.9.4. Программа заполнения таблицы <i>ПоставленныеДетали</i>	34
3. Элементы теории баз данных.....	38
3.1. Метод «сущность-связь».....	38
3.1.1. Сущности и их атрибуты	38
3.1.2. Домены	38
3.1.3. Связи между сущностями	38
3.1.4. Слабые сущности и обязательные связи.....	39
3.1.5. Документирование сущностей и связей	39
3.2. Реляционная модель данных.....	40

3.2.1. Отношения	40
3.2.2. Виды отношений	41
3.2.3. Сущности и отношения	41
3.3. <i>Реляционная алгебра</i>	41
3.4. <i>Нормализация отношений</i>	43
3.4.1. Цели проектирования базы данных	43
3.4.2. Первая нормальная форма (1НФ)	44
3.4.3. Функциональные зависимости	44
3.4.4. Вторая нормальная форма	45
3.4.5. Третья нормальная форма	45
4. Язык SQL	47
4.1. <i>Краткая история SQL</i>	47
4.2. <i>SQL в Access</i>	47
4.3. <i>Работа с таблицами</i>	49
4.3.1. Создание таблиц	49
4.3.2. Создание индекса	51
4.4. <i>Инструкция SELECT</i>	52
4.4.1. Примеры использования инструкции SELECT	53
4.5. <i>Использование SQL в VBA</i>	56
4.5.1. Классы для работы с запросами	56
4.5.2. Вывод содержимого запроса	58
4.5.3. Сохранение запроса в таблице базы данных	59
Литература	62
4.6. <i>Основная</i>	62
4.7. <i>Дополнительная</i>	62

Предисловие

В данном пособии рассматривается работа с базами данных в среде популярной программы Access. Выбор Access обусловлен тем, что с помощью этой программы можно разрабатывать базы данных различной сложности: от индивидуальной, предназначенной для использования на отдельном компьютере, до систем масштаба предприятия, работающих в сетях. Access, как одно из приложений пакета MS Office, доступна практически на любом компьютере, где установлен Office, но еще недостаточно широко используется из-за слабой подготовки пользователей. Часто задачи, эффективно и надежно решаемые с помощью Access, пытаются решать с помощью программы Excel, которая предназначена в основном для организации табличных вычислений и не ограждает от ошибок при работе с базами данных.

Access обладает развитой системой *меню* для выполнения большинства манипуляций с базами данных. Для начинающих пользователей большую помощь оказывают *мастера*, которые предлагают последовательность шагов с подробными подсказками для создания таблиц, форм, отчетов и выполнения других действий. *Конструкторы* предоставляют опытным пользователям полные возможности по созданию объектов базы данных. Все это позволяет быстро освоить работу с базами данных без изучения какого-либо языка программирования даже людьми, не являющимися профессионалами в области информационных технологий.

Рассмотрение начинается с общих вопросов, относящихся к базам данных, затем рассматривается конкретная работа с базами данных в среде Access. После того, как читатель получит конкретные навыки по работе с базами данных в среде конкретной СУБД Access, рассматривается теория реляционных баз данных, язык SQL, нестандартная обработка данных с использованием языка программирования VBA в Access.

1. Управление приложением Access

1.1. Главная кнопочная форма

Главная кнопочная форма позволяет создавать удобный интерфейс для работы с базой данных, обеспечивая быстрый доступ к таблицам, формам, отчетам и другим объектам базы данных нажатием соответствующих кнопок на кнопочной форме.

Кнопочная форма создается с помощью диспетчера кнопочных форм. Чтобы найти команду для запуска диспетчера кнопочных форм надо выполнить следующие действия.

Выполним команду **Параметры** из меню **Файл**. В окне **Параметры Access** (рис.1.1) выберем **Панель быстрого доступа** и добавим команду **Диспетчер кнопочных форм** с вкладки **Работа с базами данных**.

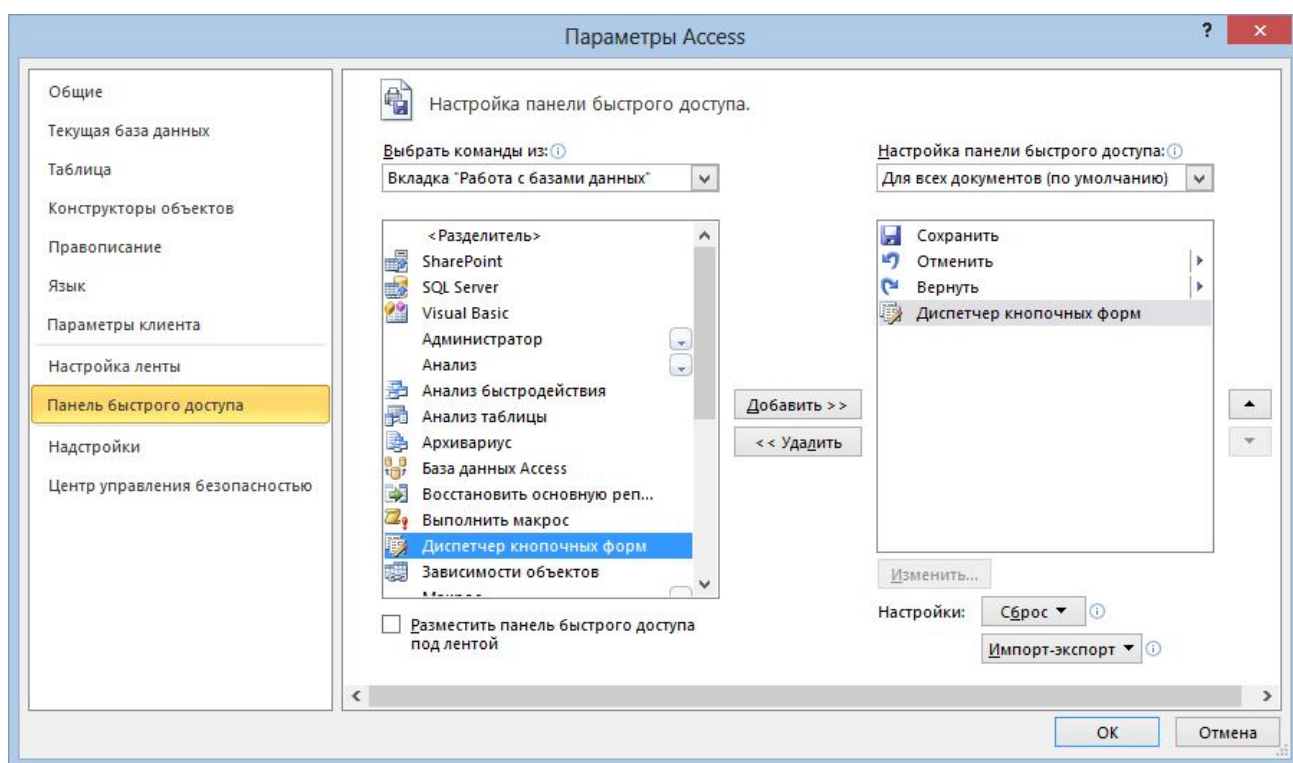


Рис. 1.1. Добавление команды на панель быстрого доступа

На рис.1.2 показано окно Access и команда **Диспетчер кнопочных форм** на **Панели быстрого запуска**.

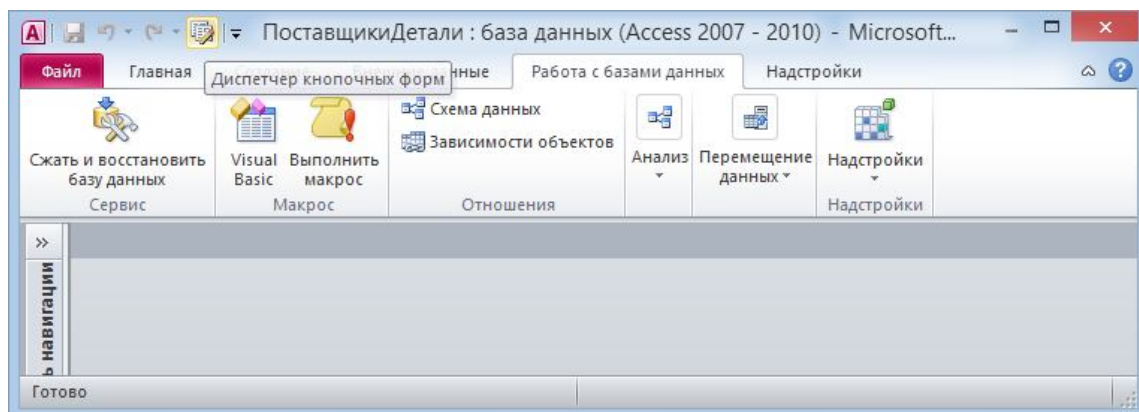


Рис. 1.2. Кнопка запуска диспетчера кнопочных форм

Выполним команду **Диспетчер кнопочных форм**. Если база данных не содержит кнопочных форм, выводится диалог, показанный на рис.1.3.

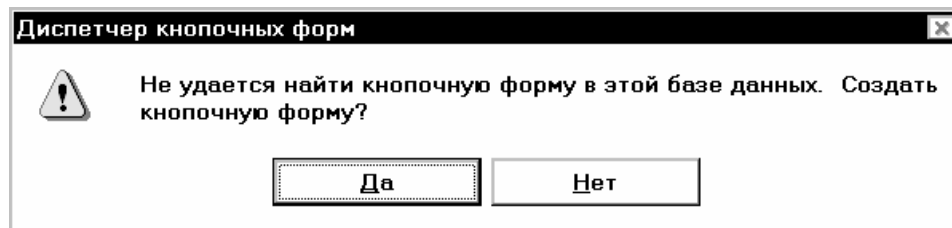


Рис. 1.3. Начало создания кнопочной формы

После нажатия кнопки **Да** появляется диалог **Диспетчер кнопочных форм** (рис. 1.4), в котором перечисляются все страницы кнопочной формы. По умолчанию создается одна страница, имя которой (**Main Switchboard** или **Главная кнопочная форма**) показывается в списке.

Кнопка **Создать** позволяет добавить на кнопочную форму еще одну страницу.

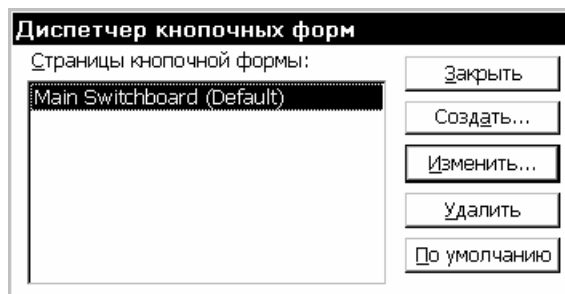


Рис. 1.4. Диалоговое окно Диспетчера кнопочных форм

Для создания кнопок на существующей странице кнопочной формы нужно нажать кнопку **Изменить**. Появится диалог **Изменение страницы кнопочной формы**. На рис. 1.5 этот диалог показан в начальном состоянии, когда еще не созданы элементы кнопочной формы. На рис. 1.7 в этом диалоге показаны созданные элементы кнопочной формы.

Для создания нового элемента кнопочной формы нужно нажать кнопку **Создать** в окне **Изменение страницы кнопочной формы** (рис. 1.5). Нажав кнопку **Изменить**, можно отредактировать существующий элемент кнопочной формы. Существующий элемент удаляется с помощью кнопки **Удалить**. Взаимное расположение элементов кнопочной формы изменяется с помощью кнопок **Вверх** и **Вниз**.

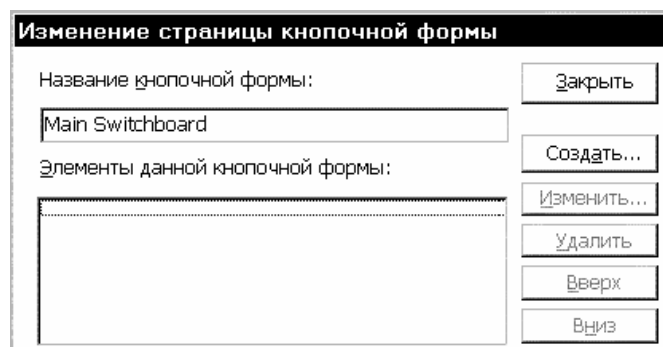


Рис. 1.5. Редактирование элементов кнопочной формы

После нажатия кнопки **Создать** вызывается окно для задания параметров нового элемента кнопочной формы (рис. 1.6).

Рис. 1.6. Создание элемента кнопочной формы

В поле **Текст** вводится надпись, поясняющая назначение создаваемой кнопки, в списке **Команда** выбирается команда, которая будет выполняться при нажатии кнопки. Если выбрана команда, требующая аргумента, появляется список для его выбора. Например, для команды **Открыть форму для изменения** список возможных аргументов будет называться **Форма**. В нем следует выбрать форму, которая будет открыта при нажатии создаваемой кнопки. По установкам, сделанным на рис. 1.6, на кнопочной форме будет создана кнопка для открытия формы *Поставщики* в режиме изменения записей таблицы *Поставщики*.

Аналогично добавляются другие кнопки на кнопочную форму (рис. 1.7).

Рис. 1.7. Список кнопок на кнопочной форме

С кнопкой *Завершить работу с БД* свяжем команду **Выйти из приложения** (рис. 1.8).

Рис. 1.8. Параметры кнопки, завершающей работу с базой данных

После включения в форму необходимых элементов кнопкой **Закрыть** закрываем окно **Изменение страницы кнопочной формы** (рис. 1.7), а затем окно **Диспетчера кнопочных форм** (рис. 1.4). В списке форм окна базы данных появится форма с названием **Switchboard (Кнопочная форма)**, а в списке таблиц появится таблица **Switchboard Items**, содержащая описания элементов кнопочной формы (рис. 1.9).

Switchboard Items : таблица					
	SwitchboardID	ItemNumber	ItemText	Command	Argument
▶	1	0	Main Switchboard		Default
	1	1	Поставщики	3	Поставщики
	1	2	Детали	3	Детали
	1	3	Поставки	3	Поставки
	1	4	Отчет о деталях	4	Детали
	1	5	Завершить работу с БД	6	
*		0			
Запись: 1 из 6					

Рис. 1.9. Таблица, описывающая элементы кнопочной формы

Для того чтобы кнопочная форма загружалась при запуске приложения, выполним команду **Файл, Параметры**, в окне **Параметры Access** в поле **Форма просмотра** выберем созданную кнопочную форму (рис. 1.10).

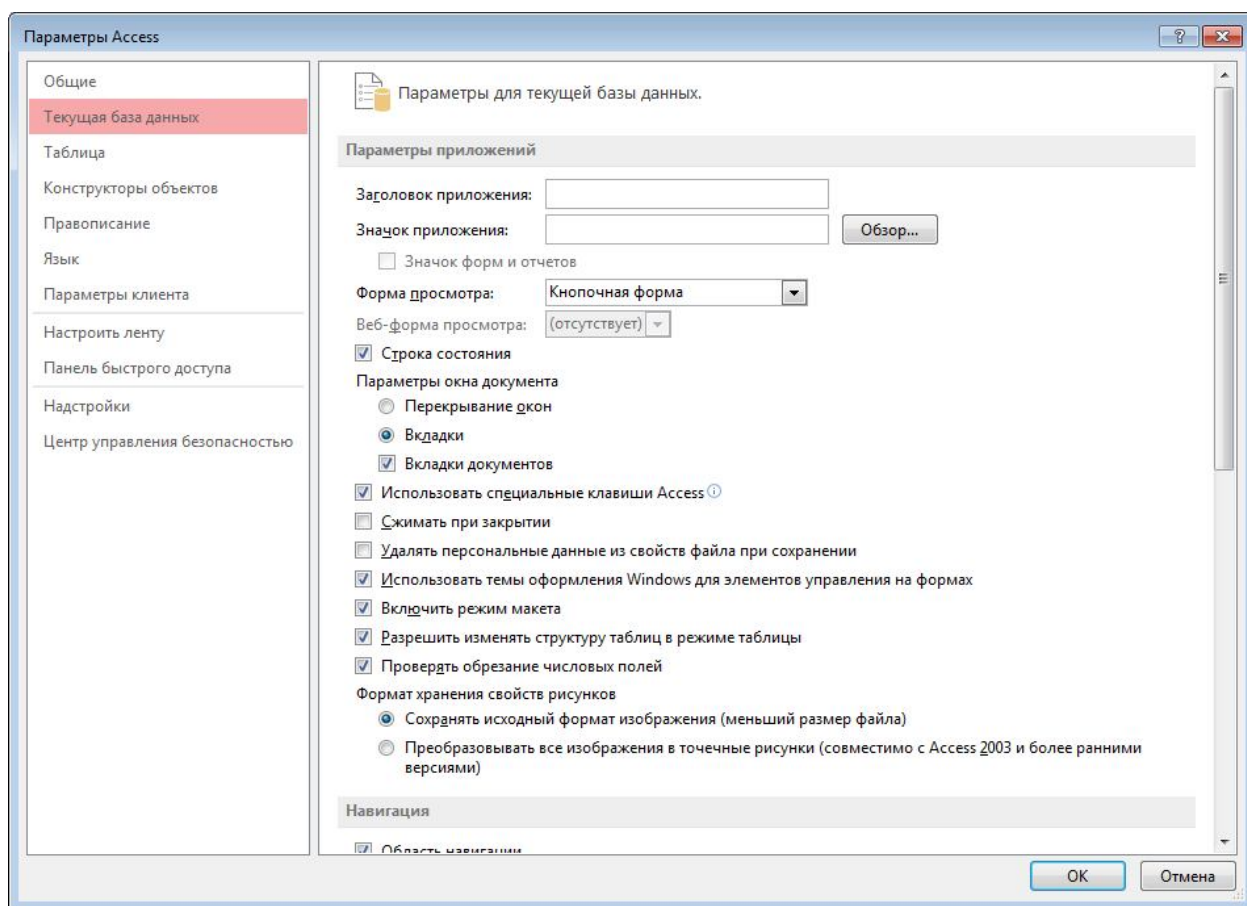


Рис. 1.10. Выбор формы, начинающей работу с базой данных

Созданная кнопочная форма показана на рис. 1.11.

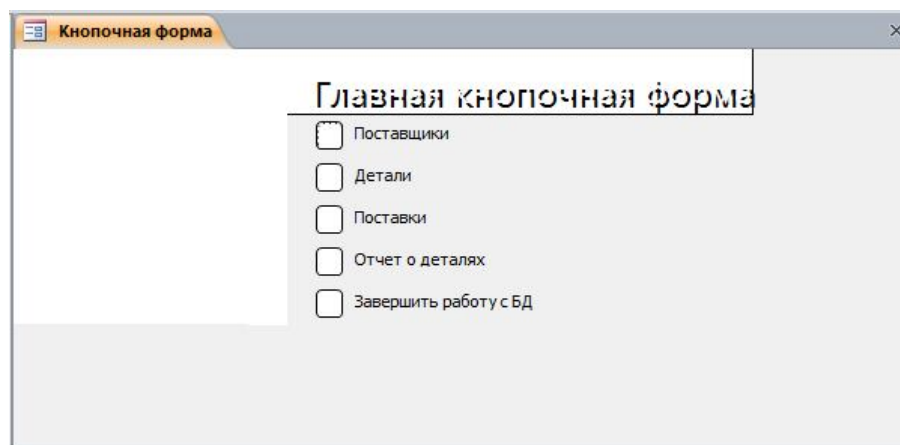


Рис. 1.11. Вид главной кнопочной формы

1.2. Анализ базы данных

Средства анализа содержимого базы данных позволяют оптимизировать структуру таблиц, исключив дублирование информации.

1.2.1. Анализ таблиц

В состав Access входят средства, с помощью которых можно выявить повторяющиеся данные в таблицах и разместить повторяющиеся сведения в новых таблицах. Соответствующий инструмент **Анализ таблиц** расположен на вкладке **Работа с базами данных** в группе **Анализ**.

Выполним команду **Анализ таблиц**. Начнет работать мастер анализа таблиц. Первые два диалога показаны на рисунках 1.12, 1.13. Они объясняют, как таблицу можно разделить на две или более для устранения избыточности данных.

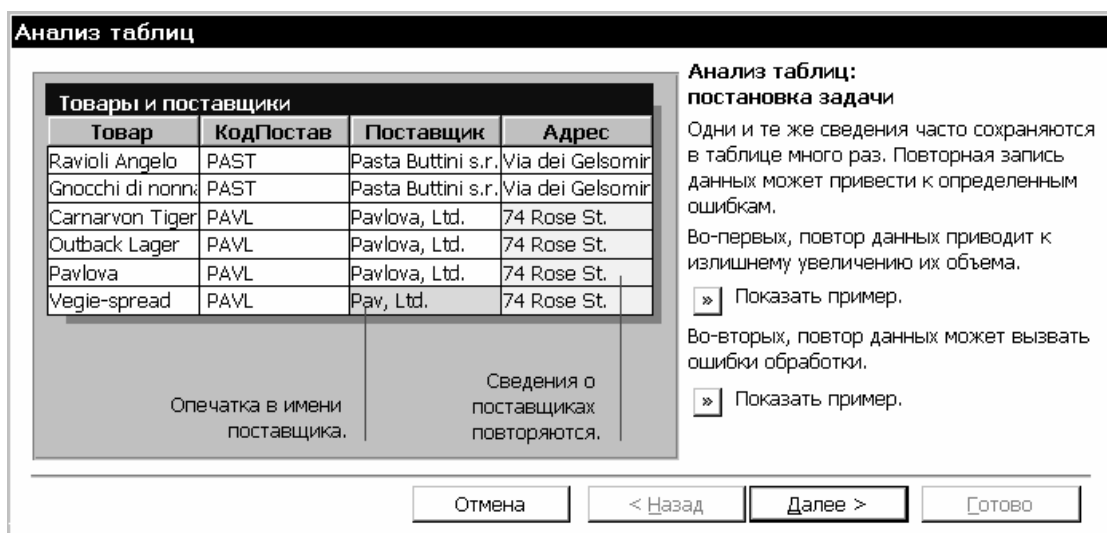


Рис. 1.12. Первый диалог мастера анализа таблиц с объяснением постановки задачи

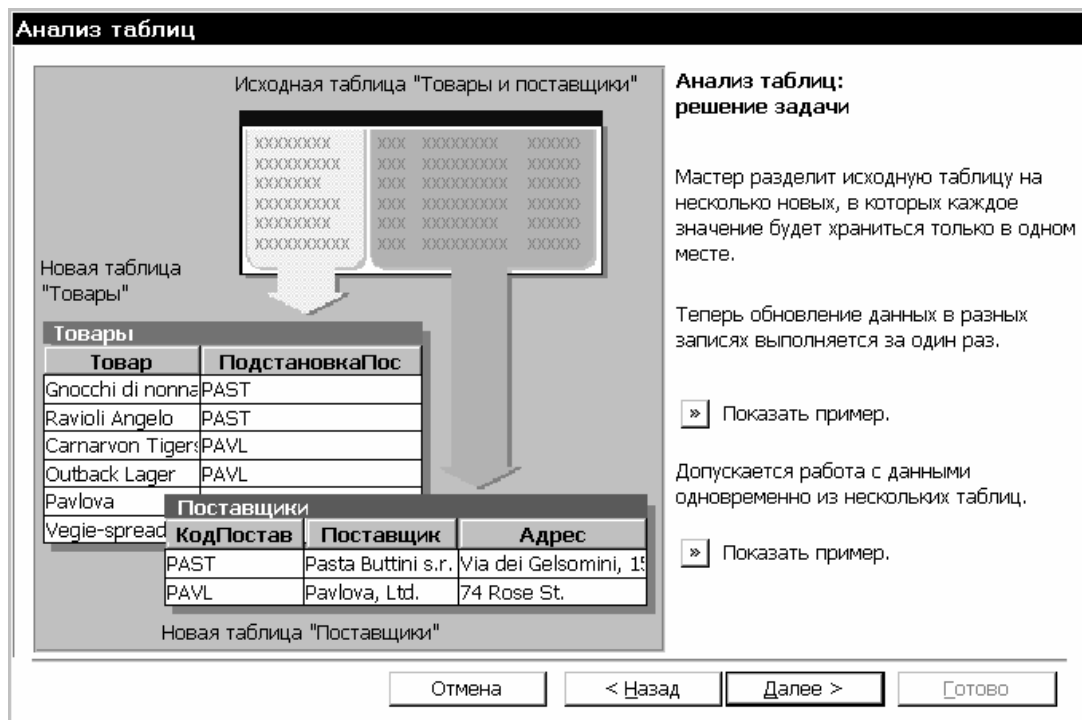


Рис. 1.13. Второй диалог мастера анализа таблиц с примером

На третьем диалоге (рис. 1.14) выбираем таблицу для анализа, например, *Поставщики*.

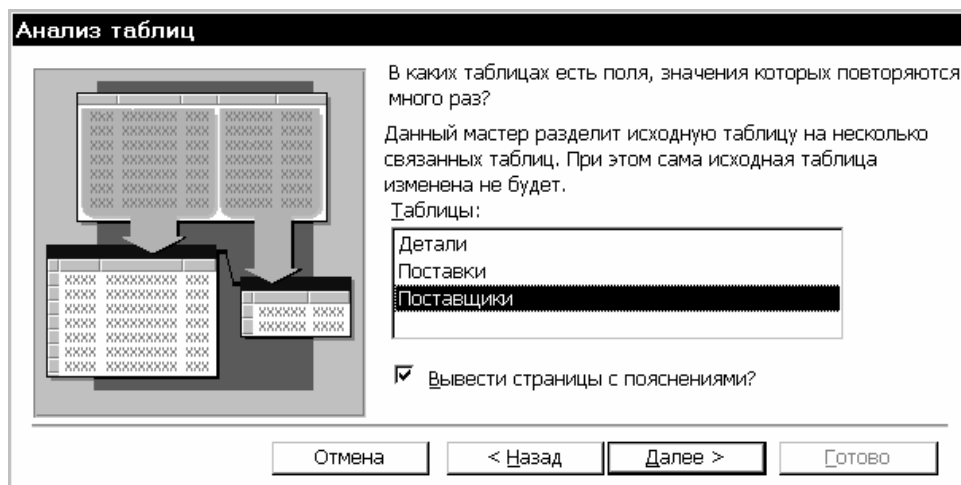


Рис. 1.14. Выбор таблицы для анализа

На четвертом диалоге (рис.1.15) можно выбрать способ разделения полей: *ручной* или *автоматический* с помощью мастера.

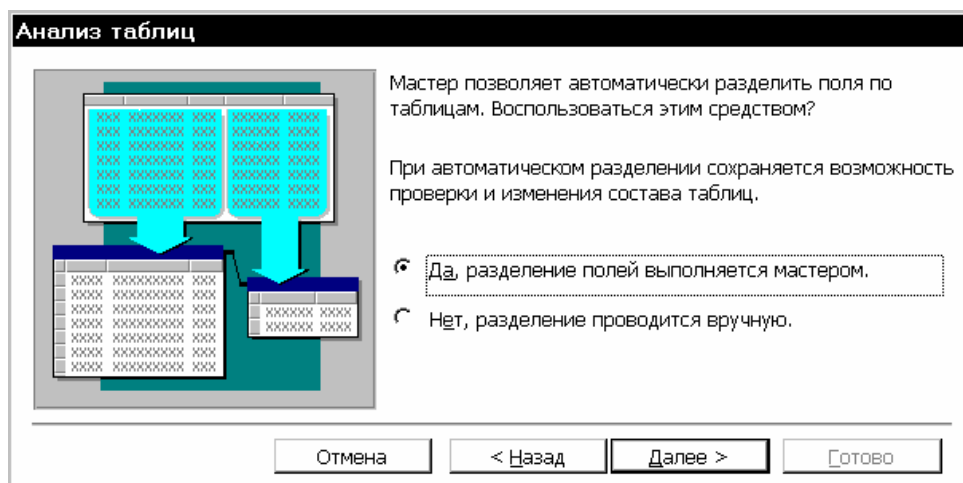


Рис. 1.15. Выбор способа разделения таблицы

Если мастер анализа таблиц решит, что таблицу разделять не следует, будет выведено информационное сообщение (рис. 1.16), на котором, однако, имеется кнопка **ОК**, нажав которую можно перейти к ручному разделению таблицы.

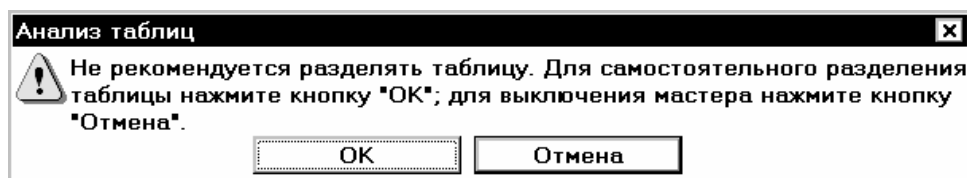


Рис. 1.16. Информация мастера анализа таблиц

Разделение таблицы производится на специальном бланке (рис. 1.17). Для преобразования поля или нескольких полей в отдельную таблицу их следует перетащить мышью из исходной таблицы наружу. На рис. 1.17 в отдельную таблицу выделено поле *Город*, так как в таблице *Поставщики* у этого поля есть повторяющиеся значения. Сразу же появляется окно для ввода имени созданной таблицы, которая по умолчанию называется **Таблица2**. Изменим это название на *Города*, а **Таблицу1** назовем *Поставщики1*.

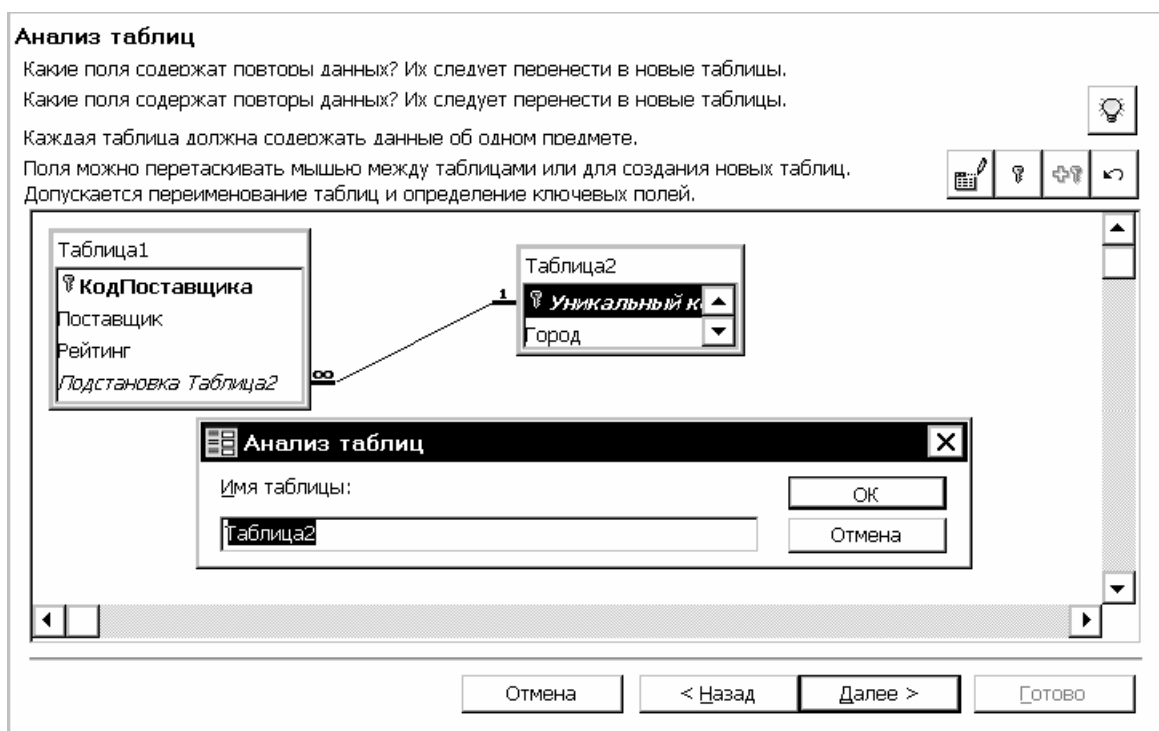


Рис. 1.17. Разделение таблицы

Ряд действий в окне **Анализ таблиц** можно выполнить с помощью кнопок:



— выводит окно с советами;



— переименовать таблицу;



— сделать поле ключевым;



— добавить ключевое поле.

Окно с перечнем советов приведено на рис. 1.18.



Рис. 1.18. Советы по разделению таблиц

После нажатия кнопки **Далее** на диалоге, показанном на рис. 1.17, появится окно с вопросом о создании запроса вместо старой таблицы (рис. 1.19). Если согласиться на создание запроса, то будет создан запрос *Поставщики*, который будет использован везде, где ранее использовалась таблица *Поставщики*, а сама таблица *Поставщики* будет названа *Поставщики_СТАРАЯ*.

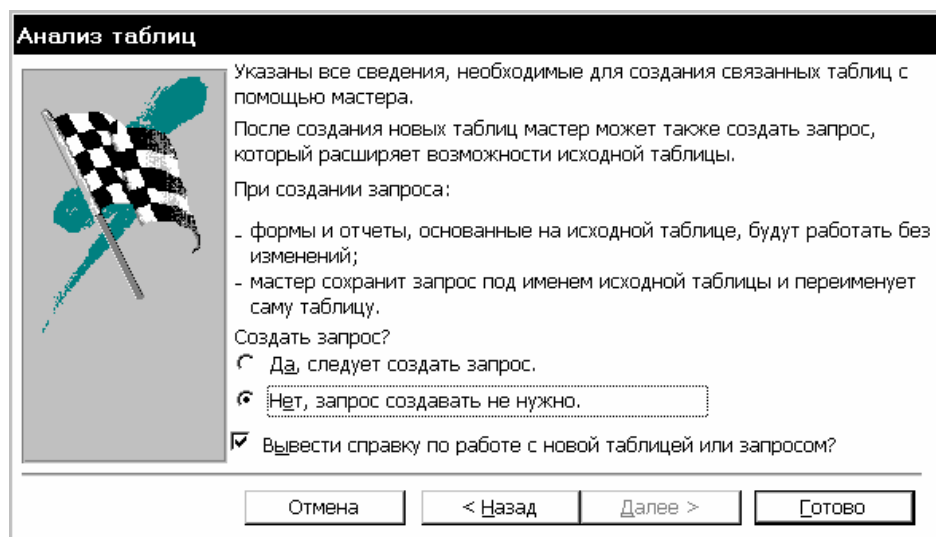


Рис. 1.19. Предложение создать запрос вместо разделенной таблицы

Откажемся от замены старой таблицы запросом. В результате будут созданы две новые таблицы (рис. 1.20). Далее можно решить, использовать ли их вместо одной старой или нет.

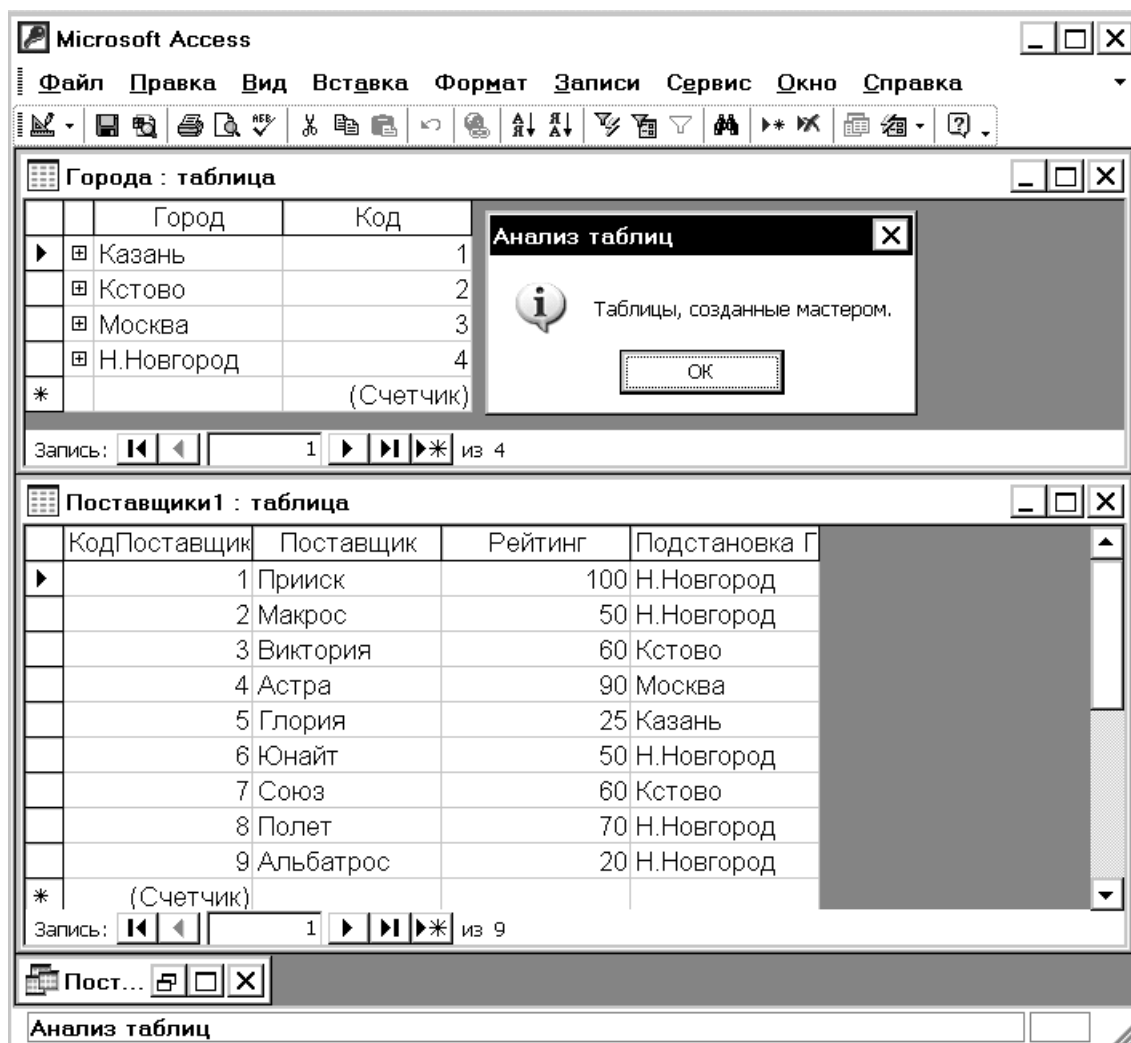


Рис. 1.20. Две таблицы, полученные из таблицы *Поставщики*

1.3. Защита базы данных

Сведения, хранящиеся в базе данных, могут содержать коммерческую или личную тайну, поэтому может возникнуть необходимость в защите данных от несанкционированного доступа. Access имеет ряд возможностей по защите данных: установка пароля, шифрование данных, создание групп пользователей с различными правами. Рассмотрим простейшую защиту БД с помощью пароля.

1.3.1. Установка пароля

Установка пароля возможна, если база данных открыта в режиме *монопольного доступа*. Для этого при открытии БД нужно выбрать режим **Монопольно** в списке кнопки **Открыть** диалога **Открытие файла базы данных** (рис. 1.21).

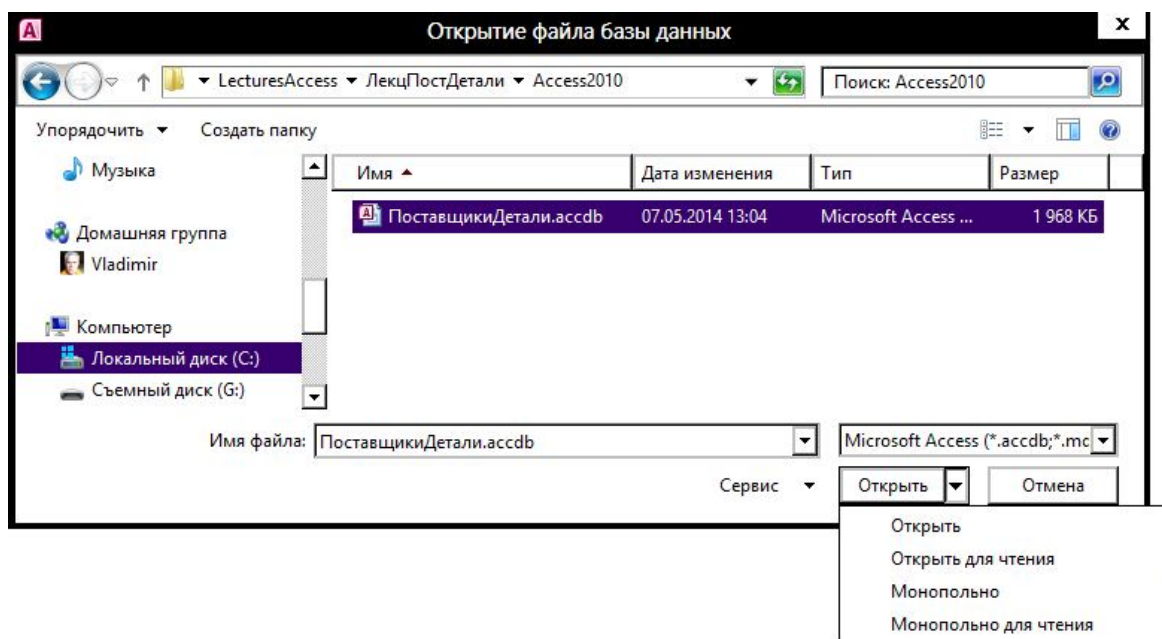


Рис. 1.21. Выбор режима открытия базы данных

Для установка пароля выполним команду **Файл, Сведения, Зашифровать паролем** (рис.1.22).

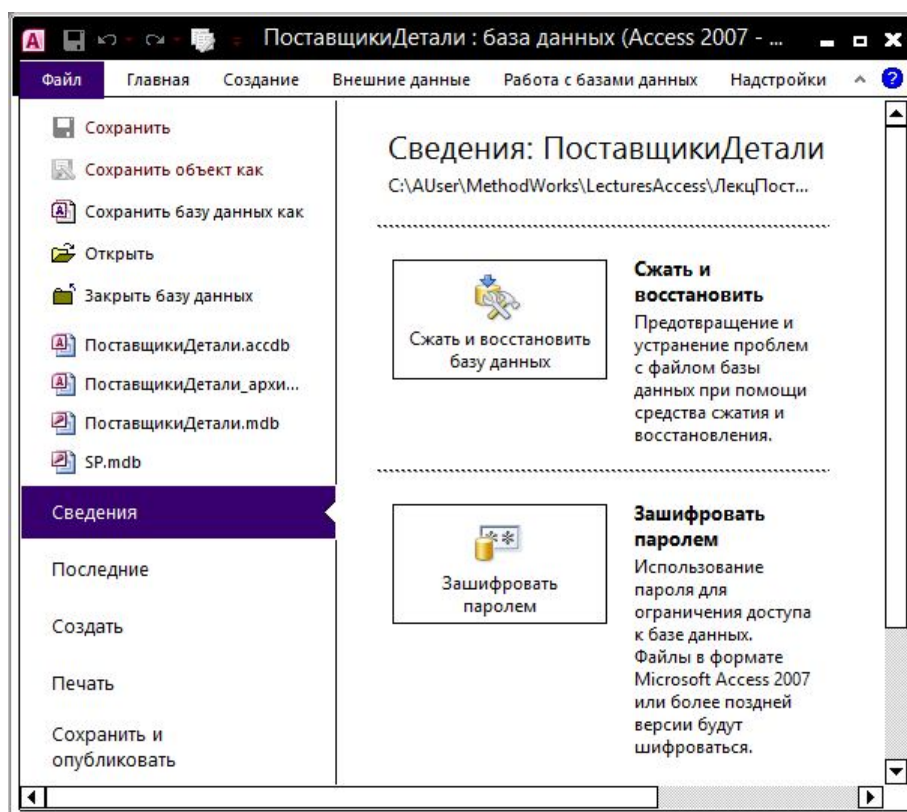


Рис. 1.22. Команда для создания пароля

Появится окно для ввода пароля (рис. 1.23), в котором надо ввести и подтвердить пароль.



Рис. 1.23. Задание пароля

После этого при каждом открытии базы данных будет предлагаться ввести пароль, рис. 1.24.

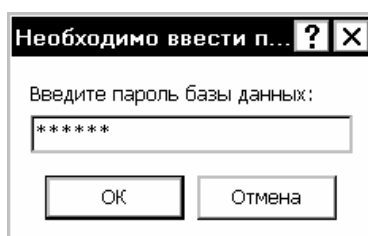


Рис. 1.24. Ввод пароля при открытии базы данных

1.3.2. Удаление пароля

Для удаления пароля базу данных следует открыть в монопольном режиме (рис. 1.21). Если пароль установлен, то в меню **Файл** в группе **Сведения** появится команда **Расшифровать базу данных** (рис.1.25).

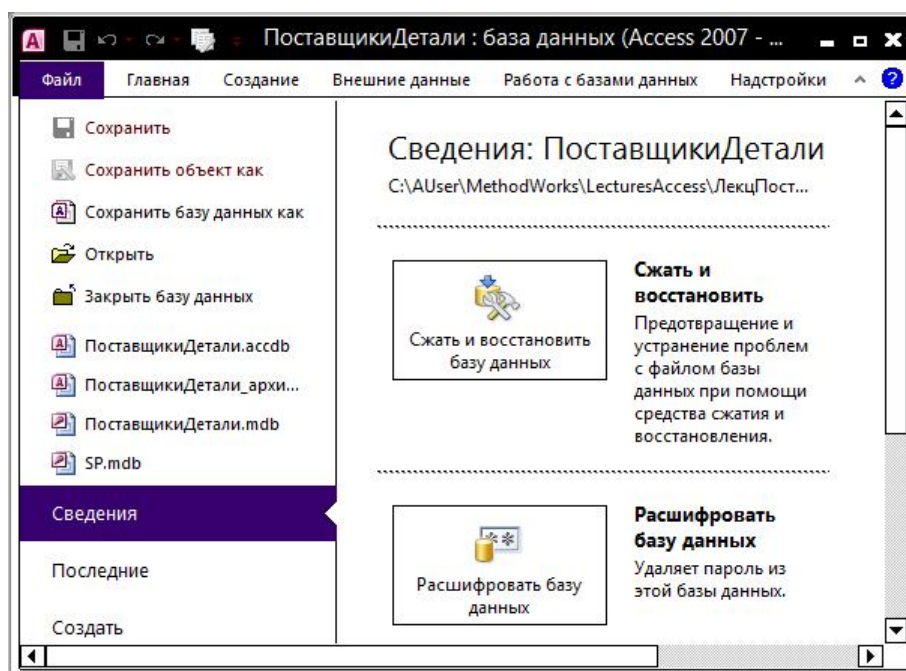


Рис. 1.25. Команда удаления пароля

Указанная команда выводит окно (рис.1.26), в котором нужно ввести старый пароль. После нажатия **ОК** пароль будет удален.

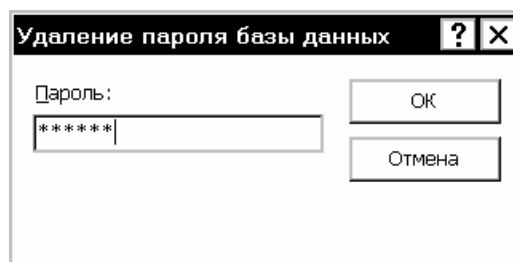


Рис. 1.26. Ввод старого пароля при его удалении

2. Модули

Объекты **Модули** Access содержат программный код на языке программирования Visual Basic for Application (VBA). Этот язык встроен в Access, он также используется в Word, Excel и некоторых других приложениях. VBA не следует путать с языком Visual Basic (VB), который является независимым средством разработки. Язык Basic (Beginners All – purpose Symbolic Instruction Code – универсальный язык для начинающих) появился в 1964 г. В 1992 г. Microsoft выпустила первую систему визуальной разработки приложений Visual Basic. В 1994 г. появился язык VBA для использования в составе других приложений.

2.1. Первая программа

Командой **Visual Basic** из группы **Макросы и код** со вкладки **Создание** загружается редактор VBA в отдельном, независимом от Access окне (рис. 2.1). Здесь может быть открыто несколько окон различного назначения, для которых родительским окном будет окно редактора. Видом среды VBA управляют команды меню **View**.

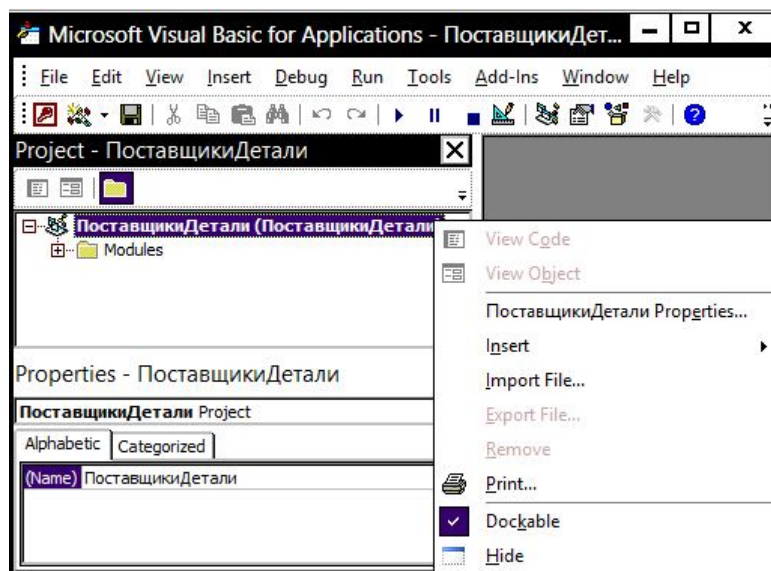


Рис. 2.1. Редактор VBA и контекстное меню для проекта

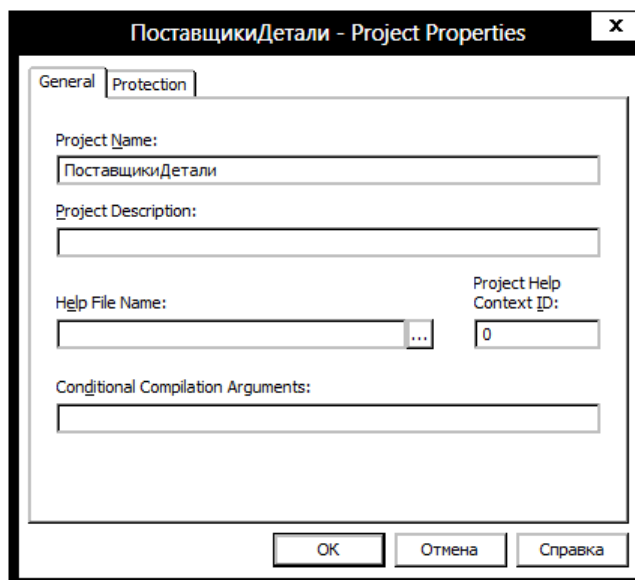


Рис. 2.2. Настройка свойств проекта

С каждой базой данных связывается *проект* VBA, структура которого показывается в окне **Project**. Командой контекстного меню **Properties** (рис. 2.1), выводится окно свойств проекта (рис.2.2), в котором можно сделать некоторые настройки, например, изменить имя проекта, задать описание проекта в поле **Project Description**, связать с проектом файл справки в поле **Help File Name**.

Новый модуль кода включается в проект, если в среде Access выполнить команду **Модуль** со вкладки **Создание**, или выполнить команду **Insert, Module** в редакторе VBA. Модуль по умолчанию будет назван **Module1**. Выполним эту команду и в появившемся окне редактирования программы (рис.2.3) введем следующий код:

```
Sub hello()
    MsgBox("Здравствуй, мир!")
End Sub
```

Здесь слово **Sub** начинает подпрограмму, слова **End Sub** завершают подпрограмму. Стандартная функция **MsgBox** создает на экране окно и выводит в нем текст, передаваемый ей в качестве аргумента.

В состав модуля могут входить несколько подпрограмм.

Выполним команду **File, Save** (рис.2.3) и сохраним модуль под именем **Module_Hello**.

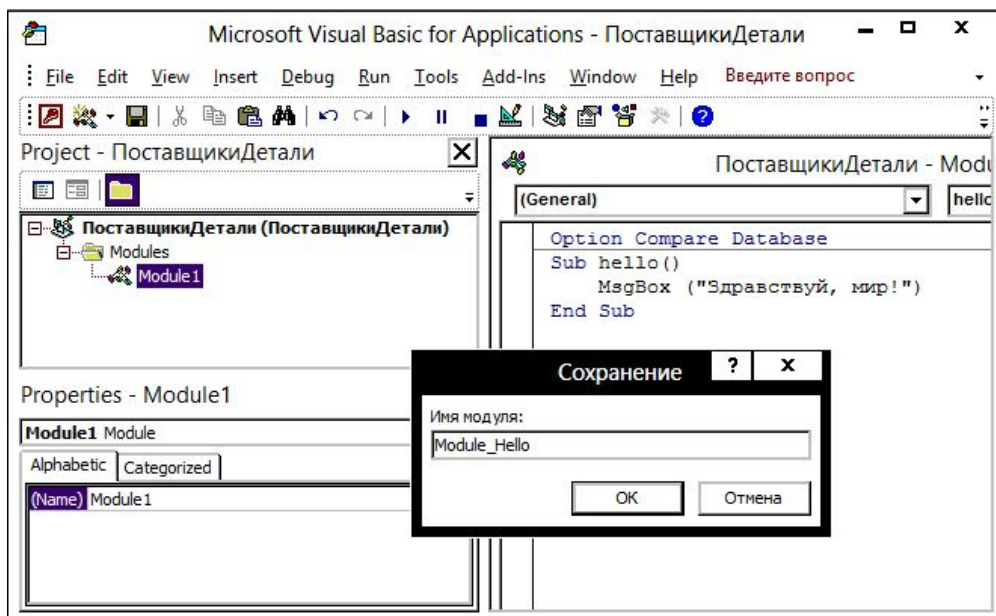


Рис. 2.3. Сохранение модуля

Для запуска программы выполним команду **Run**, **Run Sub/UserForm** или просто нажмем **F5**. Результат работы программы показан на рис.2.4.

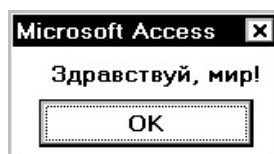


Рис. 2.4. Окно работающего программного модуля

Модуль из проекта удаляется командой **File, Remove** или такой же командой контекстного меню.

2.2. Краткий обзор языка VBA

2.2.1. Алфавит и лексика

Алфавит включает большинство символов кодовой таблицы, в том числе и русские буквы, причем русские буквы можно использовать при образовании имен объектов программы. Большие и малые буквы считаются одинаковыми.

В каждой строке программы обычно размещается одна *инструкция* (оператор). Длинные инструкции можно размещать в нескольких строках, используя в качестве знака переноса символ подчеркивания (_). Несколько инструкций можно размещать в одной строке текста, разделяя их двоеточием (:).

Комментарии к программе начинаются апострофом (') и завершаются концом строки.

2.2.2. Ключевые слова

Язык включает ключевые слова, которые имеют заранее predeterminedный смысл. Их нельзя использовать для обозначения переменных, констант, других объектов программы. Например, ключевыми словами являются названия типов данных: Integer, String.

2.2.3. Операции

Арифметические операции обозначаются общепринятым образом:

+ сложение,
- вычитание,
* умножение,
/ деление.

Имеются еще арифметические операции:

$a \wedge b$ возведение a в степень b ,
 $a \setminus b$ целая часть от деления a на b ,
 $a \text{ Mod } b$ остаток от деления a на b .

2.2.4. Типы данных

VBA поддерживает следующие типы данных:

Byte, Integer, Long	– целые, размером 1, 2 и 4 байта;
Single, Double	– числа с плавающей точкой размером 4 и 8 байт;
Currency	– денежный с 4-мя цифрами после десятичной точки;
Decimal	– любое число до 28 цифр;
Boolean	– логический со значениями True и False;
Date	– дата от 01.01.0100 до 31.12.9999;
String	– строковый, длиной до 64 Кбайт;
Object	– ссылка на объект;
Variant	– вариантный, совместимый со всеми другими типами.

Если тип переменной не указан явно, считается, что это тип Variant.

2.2.5. Переменные, константы, массивы

Явное объявление простой переменной имеет вид:

Dim ИМЯ1 [As<ТИП>] [, ИМЯ2 [As<ТИП>]]...

Здесь ИМЯ1, ИМЯ2 - имена создаваемых переменных, в квадратные скобки заключены необязательные элементы.

После каждой переменной нужно указывать тип, иначе переменная получит тип Variant. Например,

Dim s As String ' Строковая переменная
Dim i As Integer, d As Double ' i - целого типа, d – плавающего двойной точности

Объявление констант имеет вид:

Const ИМЯ_КОНСТАНТЫ [As<ТИП>] = Выражение

При определении массивов размерность указывается в *круглых* скобках:

Dim ИМЯ_МАССИВА (<размер1>[,размер2>]...) [As <ТИП>]

Например,

Dim A(10, 10) As Integer ' Целочисленный двумерный массив

Индексация элементов массива начинается с 1.

2.2.6. Операторы цикла и условия

VBA имеет несколько операторов цикла.

Циклы со счетчиком объявляются с использованием ключевого слова For, например, вычислить и вывести значение факториала можно с помощью следующей подпрограммы:

```
Sub Fact20()  
    f = 1
```

```

For i = 20 To 1 Step -1    ' Цикл по убыванию
    f = f * i
Next i
MsgBox (f)
End Sub

```

Программа выдаст значение 2.43290200817664E+18.

Если шаг цикла **Step** явно не указан, он принимается равным 1.

Существуют два вида *циклов с условием*, отличающихся местом расположения условия. Цикл с *предусловием* записывается в виде:

```

Do While Условие
    [Блок операторов]
Loop

```

Цикл с *постусловием* имеет вид:

```

Do
    [Блок операторов]
Loop While Условие

```

Данные циклы повторяются до тех пор, пока условие остается *истинным*.

Если ключевое слово **While** заменить на **Until**, то циклы будут повторяться, пока условие остается *ложным*.

Существует также простой цикл с предусловием:

```

While Условие
    [Блок операторов]
Wend

```

Условный оператор применяется для выбора одного из нескольких вариантов выполнения программы. Его синтаксис следующий:

```

If условие Then
    [Блок_Операторов1]
Else
    [Блок_Операторов2]
End If

```

2.2.7. Процедуры и функции

В виде процедур оформляются фрагменты программы, выполняющие некоторую самостоятельную часть вычислений. Процедуры оформляются следующим образом:

```

[Private|Public] Sub ИмяПроцедуры([Список_аргументов])
    [Блок_Операторов]
End Sub

```

Здесь вертикальная черта разделяет альтернативные варианты.

Ключевое слово **Private** задает в качестве области видимости процедуры только текущий модуль, **Public** делает областью видимости все модули проекта.

По умолчанию аргументы передаются *по ссылке*, но можно задать это явно с помощью ключевого слова **ByRef**, например, следующая процедура меняет местами значения двух переменных:

```

Sub Change (ByRef a As Integer, ByRef b As Integer)
    Dim Tmp As Integer
    Tmp = a: a = b: b = Tmp
End Sub

```

Двоеточие разделяет инструкции программы, расположенные в одной строке.

Для передачи аргументов *по значению* используется ключевое слово **ByVal**.

Функции – это подпрограммы, возвращающие значение. Внутри функции должен быть оператор присваивания имени функции некоторого значения. Далее приведен текст функции, возвращающей максимальное значение своих аргументов:

```
Public Function Max(ByVal A As Integer, ByVal B As Integer) As Integer
    If A > B Then
        Max = A      ' Имени функции присваивается значение
    Else
        Max = B
    End If
End Function
```

VBA имеет обширную встроенную библиотеку процедур и функций, которые обеспечивают ввод и вывод, работу с файлами, преобразование и проверку типов, работу со строками, датами и временем, работу с базами данных и другие.

2.3. Среда разработки

2.3.1. Состав среды

Среда разработки предоставляет удобные средства для создания программ. Она состоит из нескольких окон, в которых могут отображаться различные сведения о создаваемой программе. Все окна расположены внутри родительского окна редактора VBA, они имеют собственные элементы управления и могут свободно располагаться на экране. Вид среды после создания модулей, о которых шла речь выше, показан на рис.2.5.

Настройка среды производится командами меню **View**. Рассмотрим эти команды и открываемые ими окна.

Команда **View, Object Browser** выводит окно просмотра объектов, в котором можно просмотреть состав всех встроенных классов, а также типов и классов пользователя, например, на рис.2.5 в этом окне виден состав класса **ClassPers**.

Команда **View, Project Explorer** выводит окно **Project**, содержащее иерархическое дерево проектов приложения и модулей этих проектов с их элементами. Здесь имеются папки, из которых можно выбрать нужный объект: **Modules** – содержит модули пользователя, **Class Modules** – модули классов, **References** – ссылки на объекты из внешних библиотек и список модулей этих библиотек.

В окне **Properties**, которое выводится по команде **View, Properties Window**, показываются свойства выбранного элемента проекта, например, на рис.2.5 в этом окне видны свойства модуля класса **ClassPers**. Значения свойств в окне **Properties** можно задавать или изменять.

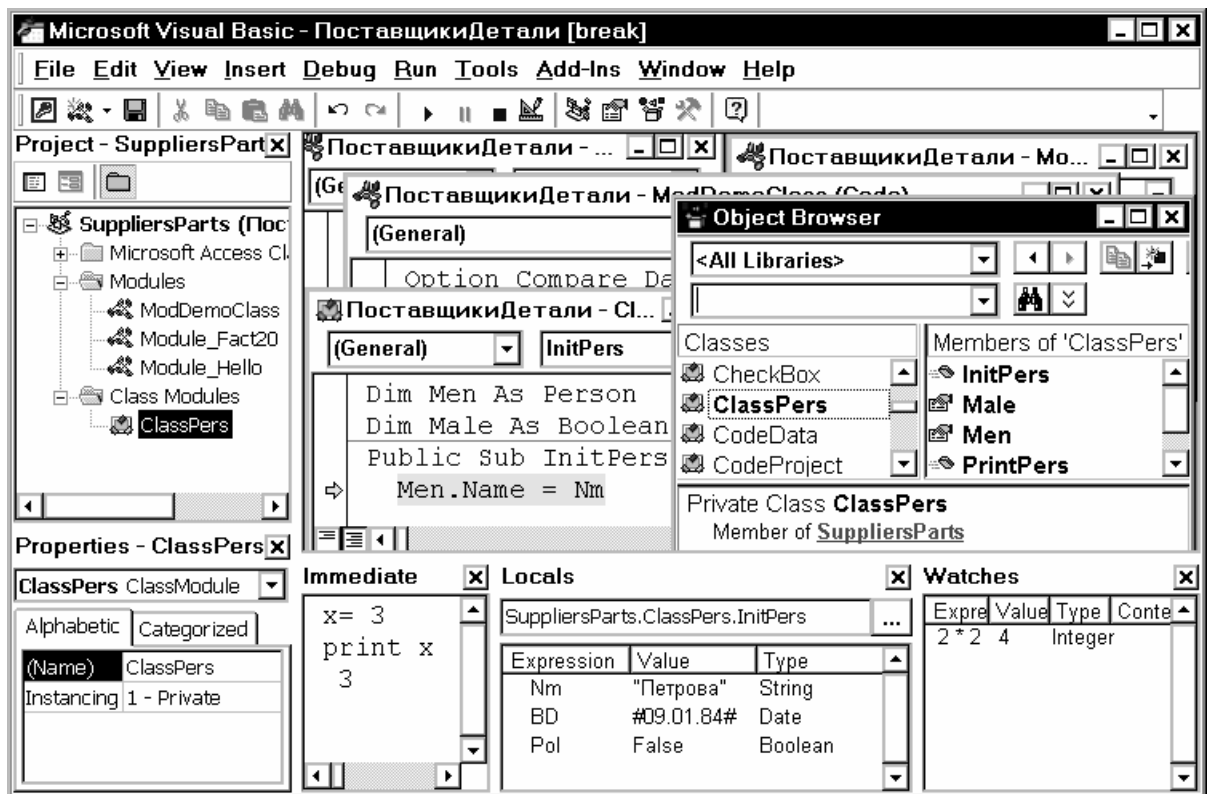


Рис. 2.5. Среда разработки VBA

Окно кода текущего модуля вызывается командой **View, Code**.

В окне **Immediate**, вызываемом командой **View, Immediate Window**, операторы Бейсика выполняются немедленно, сразу после их ввода и нажатия **Enter**. На рис. 2.5 в этом окне выполнены два оператора: `x = 3` и `print x`, и виден также результат работы второго из них.

Окно **Locals**, выводимое по команде **View, Locals Window** удобно при отладке программ: в нем автоматически показываются значения локальных переменных текущей процедуры. На рис. 2.5 в этом окне видны значения локальных переменных процедуры `InitPers`.

Окно **Watches** выводится командой **View, Watch Window**. В нем можно наблюдать значения переменных и выражений во время выполнения программы. Выражения для просмотра добавляются командой **Debug, Add Watch** или такой же командой контекстного меню. В этом окне на рис. 2.5 показано значение выражения `2*2`.

Для перехода в окно кода какого-либо модуля нужно выбрать этот модуль в окне **Project** и нажать **F7** или выполнить команду меню **View, Code**. В окне кода пишется текст программы.

2.4. Отладка

Команды отладки находятся в меню **Debug**. Перечислим эти команды:

- **Step Into (F8)** – выполнение одной строки кода, если в строке стоит вызов процедуры или функции происходит вход в них;
- **Step Over (Shift+F8)** – выполнение строки кода без захода в процедуру и функцию;
- **Step Out (Ctrl+Shift+F8)** – выполнение всех операторов процедуры и выход из нее;
- **Run to Cursor (Ctrl+F8)** – выполнение программы до точки расположения курсора.

В процессе пошагового выполнения программы после установки курсора на какой-либо переменной высвечивается её текущее значение.

При поиске ошибок в программе полезно отображать в окне просмотра **Watches** текущие значения используемых переменных.

2.5. Управление структурой проекта

Добавление новых компонентов в проект производится командами меню **Insert**:

- **Module** – добавляет в проект новый модуль;
- **Class Module** – добавляет в проект новый *модуль класса*;
- **Procedure** – в состав текущего модуля добавляется заготовка новой процедуры, команда доступна, если активно окно кода;
- **File** – позволяет вставить в окно кода содержимое внешнего файла.

Удаление какого-либо компонента из состава проекта производится командой **File, Remove**, или такой же командой контекстного меню.

2.6. Справка

Справку по какому-либо элементу VBA можно получить, установив на нем курсор и нажав клавишу **F1**. В справке можно найти полное описание VBA, но на английском языке.

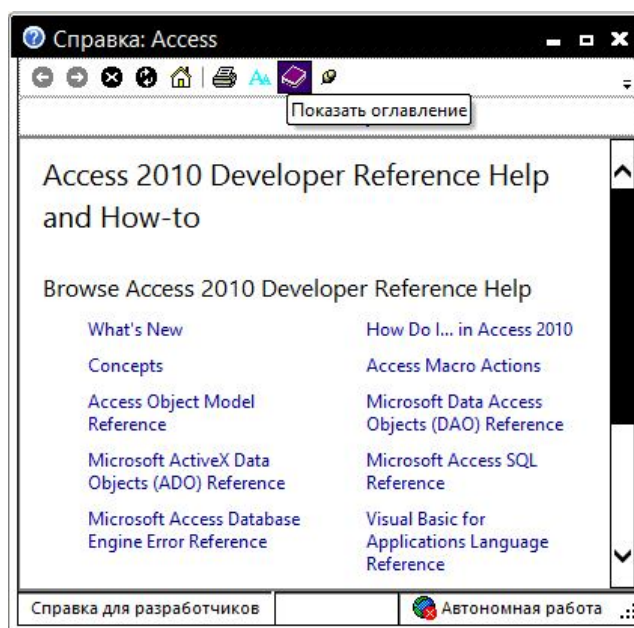


Рис. 2.6. Окно справочной системы

В окне справки можно вывести оглавление справочной системы, нажав соответствующий значек (рис.2.6), с помощью которого легко найти нужный раздел.

2.7. Классы и объекты

2.7.1. Объявление типов

От ранних версий Бейсика VBA отличается возможностью объявлять пользовательские типы данных. Например, для описания сведений о человеке можно объявить следующий тип:

```
Public Type Person  
    Name As String    'Фамилия
```

```

    BirthDay As Date 'Дата рождения
End Type

```

Ключевое слово **Public** обеспечивает видимость объявляемого имени типа во всех модулях проекта. Расположим объявление типа **Person** в начале модуля **Module_Hello**. Далее поместим процедуру, использующую новый тип данных для определения переменных:

```

Sub DemoType()
    Dim Men As Person      ' Men – переменная типа Person
    ' Задаем значения полям переменной Men:
    Men.Name = "Иванов"    ' Значение для поля Name переменной Men
    Men.BirthDay = #12/4/1950# '4 декабря 1950 г.
    MsgBox ("Гражданин " & Men.Name & " родился " & Men.BirthDay)
End Sub

```

Для запуска конкретной подпрограммы модуля нужно поместить курсор в пределах ее кода и нажать клавишу **F5** или выполнить команду **Run, Run Sub/UserForm**. Процедура **DemoType()** выдаст окно, показанное на рис.2.7.

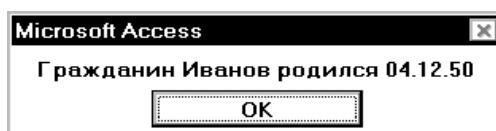


Рис. 2.7. Применение пользовательского типа

2.7.2. Понятие класса

VBA является объектно-ориентированным языком, это значит, что в нем можно работать с классами и объектами. Программирование с использованием классов и объектов называется *объектно-ориентированным программированием* (ООП).

Класс – это расширенный пользовательский тип данных. Расширение заключается в возможности включения в состав класса кроме данных, как в обычном типе (см. 2.7.1), еще процедур и функций. Данные, входящие в класс, называются *свойствами* класса, процедуры и функции, входящие в класс, называются *методами* класса. Объединение данных и процедур (функций) в одном типе называется *инкапсуляцией*.

Объект – это переменная, имеющая тип некоторого класса.

VBA имеет большую библиотеку встроенных классов, позволяющих работать с формами, отчетами, макросами, элементами управления.

В VBA класс оформляется в виде *модуля класса*. Имя такого модуля является одновременно *именем класса*. Для создания модуля класса нужно выполнить команду **Insert, Class Module**. Все *переменные*, объявленные в модуле класса, будут *свойствами* класса, все *процедуры и функции*, объявленные в модуле класса, станут *методами* класса.

2.7.3. Пример класса

Разработаем класс **ClassPers** для моделирования набора сведений о личности человека, используя при этом ранее разработанный тип **Person**.

Выполним команду **Insert, Class Module**, при этом будет создан новый класс с именем **Class1**. Изменим имя класса, введя его новое имя в строке **Name** окна **Properties** (рис. 2.8). Если окна свойств нет на экране, его можно вывести командой меню **View, Properties** или клавишей **F4**.

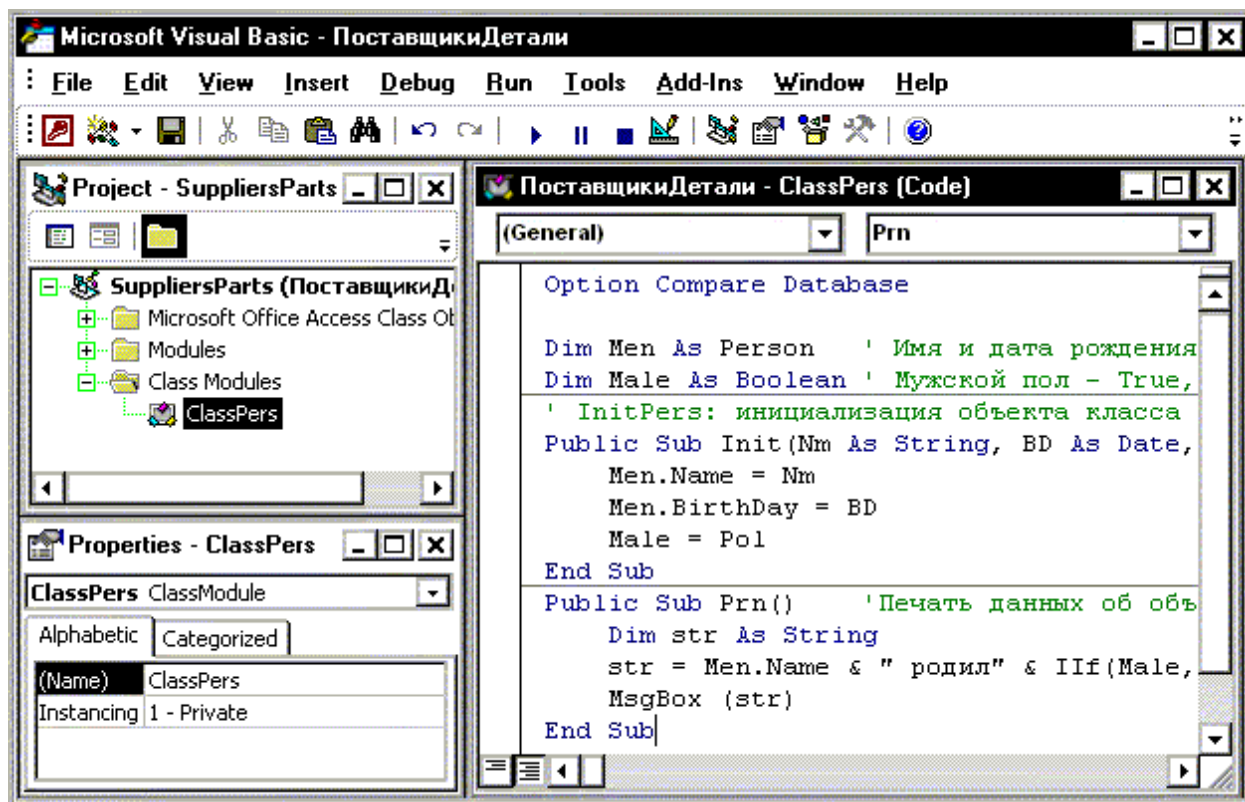


Рис. 2.8. Ввод имени класса

В окне кода введем следующий текст:

```
Dim Men As Person          ' Имя и дата рождения
Dim Male As Boolean        ' Мужской пол – True, женский – False

' Init: инициализация объекта класса
Public Sub Init(Nm As String, BD As Date, Pol As Boolean)
    Men.Name = Nm
    Men.BirthDay = BD
    Male = Pol
End Sub

Public Sub Prn()            ' Печать данных об объекте
    Dim str As String
    str = Men.Name & " родил" & IIf(Male, "ся ", "ась ") & Men.BirthDay
    MsgBox (str)
End Sub
```

В классе **ClassPers** свойствами класса являются **Men** типа **Person** и **Male** типа **Boolean**. Процедуры **Init()** и **Prn()** являются методами класса **ClassPers**.

Поскольку членом класса **ClassPers** является свойство **Men** типа **Person**, можно говорить, что тип **Person** *встроен* в класс **ClassPers**.

Создадим обычный модуль **ModDemoClass** для испытания созданного класса, в котором напомним следующую процедуру:

```
Sub DemoClass()
    Dim Stud As New ClassPers      ' Stud – объект класса ClassPers
    Stud.Init "Петрова", #1/9/1984#, False ' Инициализация объекта
    Stud.Prn                       ' Печать объекта
End Sub
```

При запуске данного модуля, выводится окно (рис. 2.9):

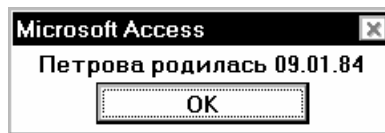


Рис. 2.9. Использование пользовательского класса

Обратим внимание на то, что в программах на VBA используется американский формат даты «месяц/день/год», а в окнах сообщений применяется формат даты, установленный в настройках Windows, в данном случае «день.месяц.год».

2.7.4. Объекты и ссылки на объекты

Обратим внимание на инструкцию создания объекта Stud:

```
Dim Stud As New ClassPers
```

Ключевое слово **New** означает, что под объект выделяется память, а имя **Stud** указывает адрес выделенной памяти.

Если ключевое слово **New** опустить, объект создан не будет, то есть память под объект выделена не будет, а будет создана только *ссылка* соответствующего типа, которую позднее можно связать с реальным объектом. Размер ссылки равен размеру адреса.

Для связывания ссылки с объектами используется оператор **Set**.

Напишем процедуру, демонстрирующую использование ссылок.

```
Sub DemoRef()
    Dim Stud As New ClassPers      ' Создание объекта класса
    Stud.Init "Петрова", #1/9/1984#, False
    Dim Ref As ClassPers           ' Ref - ссылка на объект
    Set Ref = Stud                 ' Связывание ссылки с объектом
    Ref.Prn                       ' Использование ссылки
    Dim Stud_1 As New ClassPers    ' Еще один объект класса ClassPers
    Set Stud_1 = Stud             ' Присваивание объектов
    Stud_1.Prn                    ' Использование копии объекта
End Sub
```

Данная процедура дважды выведет окно (рис. 2.9), сначала через ссылку **Ref**, затем через копию объекта **Stud_1**.

Простое присваивание вида **Ref = Stud** или **Stud_1 = Stud** для ссылок на объекты и для объектов классов является ошибкой.

2.7.5. Иерархия классов

Используя разработанные классы, можно создавать новые классы используя встраивание. Встраивание фактически уже было использовано при построении класса **ClassPers**, членом которого является объект **Men** типа **Person**. Приведем еще пример встраивания. Создадим новый класс **Book** (книга), членом которого будет объект **Author** (автор) класса **ClassPers**. Для этого создадим новый модуль класса командой **Insert, Class Module**. Вновь созданный класс назовем **Book** и введем нижеследующий код.

```
' Класс Book
' Свойства класса
Dim Author As ClassPers    ' Автор книги
Dim Title As String        ' Название книги

' Методы класса

' Init: инициализация объекта класса
```

```

Public Sub Init(Ttl As String, Atr As ClassPers)
    Set Author = Atr
    Title = Ttl
End Sub

Public Sub Prn()
    Author.Prn
    MsgBox ("Он написал " & Title)
End Sub

```

В метод Init() передаются в качестве аргументов строка Ttl с названием книги и объект Atr класса ClassPers, содержащий сведения об авторе.

Обратим внимание, что внутри метода Prn() класса Book вызывается метод с таким же именем для вложенного объекта Author.

В модуле ModDemoClass напомним следующую процедуру, использующую класс Book.

```

Public Sub Demo_Hierarchy()
    ' Объект класса ClassPers
    Dim Tolstoy As New ClassPers
    ' Инициализация объекта - автора
    Tolstoy.Init "Толстой Л.Н.", #9/9/1828#, True
    ' Объект класса Book
    Dim WarAndWorld As New Book ' Война и мир
    ' Инициализация объекта - книги
    WarAndWorld.Init "Война и мир", Tolstoy
    WarAndWorld.Prn
End Sub

```

Эта процедура выводит окна, показанные на рис. 2.10.

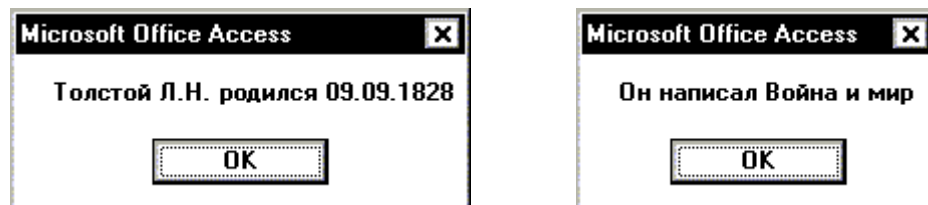


Рис. 2.10. Вывод процедуры Demo_Hierarchy

Свойства и методы класса ClassPers становятся свойствами и методами класса Book.

2.7.6. Скрытие данных

По умолчанию все члены класса являются скрытыми. Это значит, что из какой-либо внешней по отношению к классу процедуры обращаться к ним нельзя. Например, рассмотрим следующую процедуру модуля ModDemoClass, внешнего по отношению к модулям классов.

```

Public Sub Demo_Private ()
    Dim Tolstoy As New ClassPers
    Tolstoy.Male = True
End Sub

```

При ее выполнении появляется ошибка компиляции, показанная на рис. 2.11.

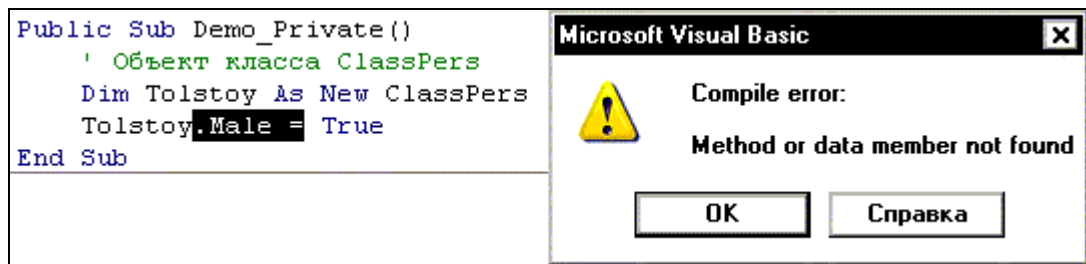


Рис. 2.11. Ошибка при обращении к закрытому члену класса

Чтобы сделать член класса открытым, он объявляется с ключевым словом **Public**. В наших классах **ClassPers** и **Book** открытыми членами класса являются методы **Init()** и **Prn()**. Если объявить в классе **ClassPers** свойство **Male** как открытое:

```
Public Male As Boolean
```

то ошибки доступа, показанной на рис. 2.11 не будет.

Методы класса имеют свободный доступ к членам своего класса.

Интересно, что несмотря на то, что все члены встроенного класса **ClassPers** являются в то же время и членами класса **Book**, методы класса **Book** не могут обратиться к закрытым членам класса **ClassPers**. Попробуем включить в состав класса **Book** метод, выводящий полные сведения о книге и ее авторе:

```
Public Sub PrnAll() 'Печать данных об объекте
    MsgBox (Author.Men.Name & " написал " & Title) ' Вывод данных о книге
End Sub
```

При вызове данного метода возникнет ошибка доступа к закрытому члену класса, подобная показанной на рис. 2.11.

2.8. Полиморфизм

Термин *полиморфизм* происходит от греческого слова *polymorphos* – многообразный. Полиморфизм в биологии – это наличие в пределах одного вида резко отличных по облику особей, не имеющих переходных форм, например, половой полиморфизм. В физике термином полиморфизм обозначают наличие несольких форм одного и того же вещества, резко отличающихся по свойствам, например графит и алмаз. В объектно-ориентированном программировании термин полиморфизм применяют для обозначения различного поведения объектов в разных местах программы.

В VBA есть встроенный тип **Object**, с помощью которого можно создавать ссылки, которые могут быть связаны с объектами различных типов. Поэтому можно говорить, что при использовании ссылок типа **Object**, реализуется полиморфное поведение объекта. Рассмотрим следующую процедуру.

```
Public Sub Demo_Polymorphism()
    Dim RefToClass As Object ' Ссылка на объект некоторого класса

    ' Два объекта класса ClassPers
    Dim Tolstoy As New ClassPers
    Dim Turgenev As New ClassPers
    ' Инициализация объектов
    Tolstoy.Init "Толстой Л.Н.", #9/9/1828#, True
    Turgenev.Init "Тургенев И.С.", #10/3/1818#, True

    ' Два объекта класса Book
    Dim WarAndWorld As New Book ' Война и мир
```

```
Dim FathersAndChildren As New Book ' Отцы и дети
WarAndWorld.Init "Война и мир", Tolstoy
FathersAndChildren.Init "Отцы и дети", Turgenev
```

```
' Использование ссылки на объект общего вида
Set RefToClass = Tolstoy
RefToClass.Prn
Set RefToClass = FathersAndChildren
RefToClass.Prn
End Sub
```

В момент работы программы, когда ссылка RefToClass связана с объектом Tolstoy, инструкция

```
RefToClass.Prn
```

выводит окно, показанное на рис. 2.12, а после связывания этой ссылки с объектом FathersAndChildren та же инструкция выводит два окна (рис. 2.13)

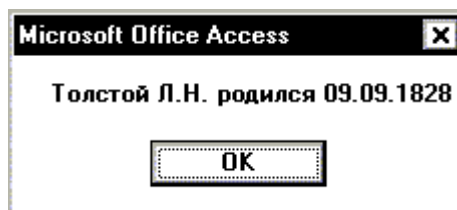


Рис. 2.12. Результат первого выполнения инструкции RefToClass.Prn

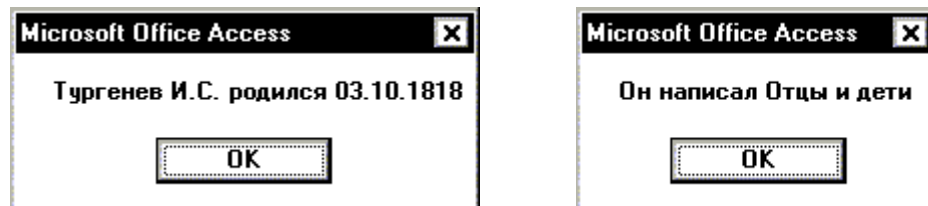


Рис. 2.13. Результат второго выполнения инструкции RefToClass.Prn

Таким образом, одна и та же инструкция программы в разные моменты времени работы программы дает разные результаты, что и является полиморфизмом.

2.8.1. Оператор With

Для доступа к членам класса приходится использовать составные имена, включающие имя объекта. Часто это приводит к длинным конструкциям, которые неудобно вводить. Оператор With создает блок, в котором можно обращаться к членам класса непосредственно по их именам. Данный оператор записывается в виде:

```
With ИмяОбъекта
' Здесь .ЧленКласса это имя члена класса
End With
```

Между With и End With выражение .ЧленКласса можно использовать вместо полного имени ИмяОбъекта.ЧленКласса.

2.9. Пример обработки данных на VBA

Существует ряд средств для работы с базами данных, которые можно использовать в программах на различных языках программирования. Эти средства объединяются в биб-

лиотеки, которые должны быть подключены, чтобы в программе можно было воспользоваться соответствующими средствами. Подключение библиотек можно осуществить программным путем или с помощью установки настроек в среде разработки. Здесь рассмотрим только последний способ подключения библиотек.

2.9.1. Подключение библиотек объектов

Библиотеки классов будут доступны в программах, если установить в среде разработки VBA соответствующую ссылку. Для создания ссылки на нужную библиотеку следует выполнить команду **Tools, References...**, в окне **References** (рис.2.14), установить флажки у нужных библиотек и нажать кнопку **OK**.

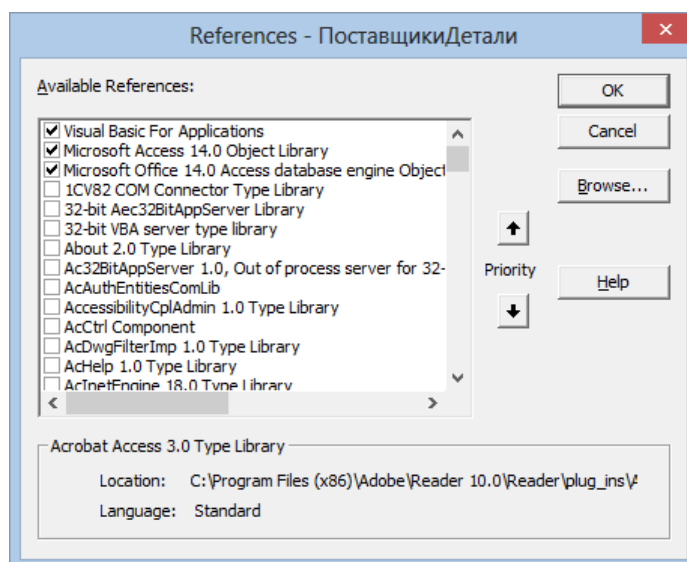


Рис. 2.14. Установка ссылок на библиотеки

В приводимой ниже программе будут использованы классы из библиотеки **Microsoft Office 14.0 Access database engine Object Library**, которую надо найти в списке и отметить.

Все доступные в программе библиотечные классы показываются в окне **Object Browser**. Например, после установки ссылки на библиотеку **Microsoft Office 14.0 Access database engine Object Library**, которая обозначается в окне **Object Browser** под кратким названием **DAO**, становится доступным класс **Database**. На рис.2.15 этот класс выделен. В правой части окна **Object Browser** показываются свойства и методы выбранного класса.

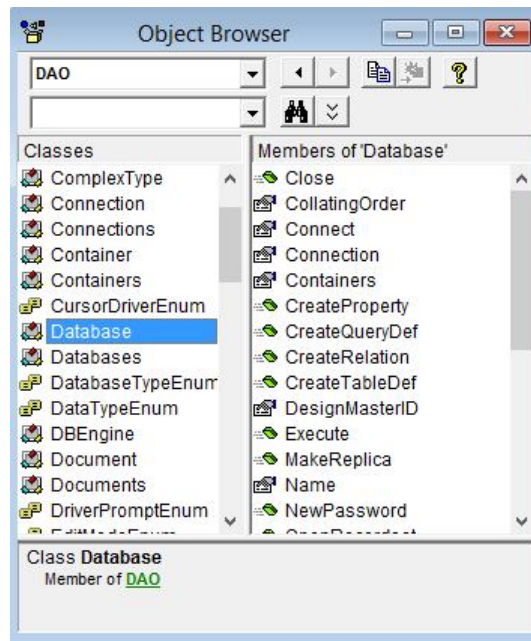


Рис. 2.15. Просмотр доступных программе библиотечных классов

2.9.2. Постановка задачи

Пусть требуется составить отчет, в каждой строке которого будет указан поставщик и перечислены через запятую названия поставленных им деталей. Отчет с подобным содержанием можно создать с помощью мастера, но он будет иметь вид, приведенный на рис. 2.16, то есть названия поставленных деталей располагаются в столбик, и нет возможности расположить их в одну строку.

<i>Поставщики1</i>	
<i>Поставщик</i>	<i>Название</i>
<i>Прииск</i>	Шпилька
	Муфта
	Винт
	Винт
	Болт
	Гайка
<i>Макрос</i>	Болт
	Гайка
<i>Виктория</i>	Болт
<i>Астра</i>	Гайка
	Муфта
	Винт
	Болт

Рис. 2.16. Стандартный отчет

Для того чтобы создать отчет нужного формата, надо сгруппировать названия всех деталей, поставленных одним поставщиком, в одну строку. Сделаем это программным путем. Решение задачи и программа приведены в следующем разделе.

2.9.3. Алгоритм решения задачи

Предварительно создадим вспомогательную таблицу *ПоставленныеДетали* с двумя полями. На рис. 2.17 эта таблица показана в режиме конструктора. Для поля *Детали* выбран максимально доступный размер 255 символов.

Рис. 2.17. Структура вспомогательной таблицы для учета поставленных деталей

Строки таблицы *ПоставленныеДетали* заполним с помощью программы, а затем обычным образом (см. главу **Ошибка! Источник ссылки не найден.**) по этой таблице создадим отчет. Текст программы приведен в следующем разделе.

2.9.4. Программа заполнения таблицы *ПоставленныеДетали*

При разработке программы следует установить ссылки на четыре стандартные библиотеки, показанные на рис.2.14. Дополнительные ссылки на библиотеки следует устанавливать только по мере необходимости, так как чем больше установлено ссылок, тем медленнее выполняется компиляция проекта.

Создадим новый модуль с названием *Mod_DeliveredParts*, в состав которого включим процедуру *FillTable_DeliveredParts()*, заносящую записи в таблицу *ПоставленныеДетали*.

В процедуре создаются две переменные строкового типа *strSupplier* и *strParts*. В первую из них заносится название поставщика, а во вторую перечень поставленных им деталей. Для этого для каждого поставщика из таблицы *Поставщики* просматриваются все записи таблицы *Поставки*, и если в ней встречается код рассматриваемого поставщика, значит он поставил некоторую деталь. По коду детали из таблицы *Поставки* находится соответствующая деталь в таблице *Детали*, и ее название добавляется в строку *strParts*. После того, как будет просмотрена вся таблица *Поставки*, сформированные строки заносятся как значения полей очередной записи таблицы *ПоставленныеДетали*.

Ниже приводится текст процедуры.

Option Compare Database

Public Sub FillTable_DeliveredParts()

Dim db As Database ' db - переменная типа Database (база данных)

Set db = CurrentDb ' Текущая база данных

' Наборы данных, представляющие

```

Dim rstSuppliers As DAO.Recordset      ' таблицу Поставщики,
Dim rstParts As DAO.Recordset          ' таблицу Детали,
Dim rstShipments As DAO.Recordset      ' таблицу Поставки и
Dim rstDeliveredParts As DAO.Recordset ' таблицу ПоставленныеДетали
    ' Присваивание наборам данных значений конкретных таблиц
Set rstDeliveredParts = db.OpenRecordset("ПоставленныеДетали")
Set rstParts = db.OpenRecordset("Детали")
Set rstSuppliers = db.OpenRecordset("Поставщики")
Set rstShipments = db.OpenRecordset("Поставки")

Dim strSupplier As String              ' Строка для названия поставщика
Dim strParts As String                 ' Строка для сохранения названий деталей

Dim lCodeSup, lCodeSupShip, lCodePart, lCodePartShip As Long ' Переменные для кодов
' Удаление всех записей из таблицы ПоставленныеДетали
With rstDeliveredParts                ' Для таблицы ПоставленныеДетали
    If Not .EOF Then                  ' Если не достигнут конец таблицы,
        .MoveFirst                    ' перейти на первую запись
        While Not .EOF                ' Пока не достигнут конец таблицы,
            .Delete                    ' удалить текущую запись,
            .MoveFirst                 ' и перейти на первую запись
        Wend
    End If
End With

With rstSuppliers                      ' Для таблицы Поставщики
    .MoveFirst                         ' Перейти на первую запись
    While Not .EOF                     ' Перебор записей в таблице Поставщики
        lCodeSup = .Fields("КодПоставщика") ' Взять код поставщика
        strSupplier = .Fields("Поставщик") ' Запомнить название поставщика
        strParts = ""                  ' Пустая строка для списка деталей
        With rstShipments              ' Для таблицы Поставки
            .MoveFirst                  ' Перейти на первую запись
            While Not .EOF              ' Перебор записей в таблице Поставки
                lCodeSupShip = .Fields("КодПоставщика") ' Код поставщика в табл. Поставки
                If lCodeSup = lCodeSupShip Then           ' Если поставщик что-то поставил
                    lCodePartShip = .Fields("КодДетали") ' Взять код детали
                    With rstParts                        ' Для таблицы Детали
                        .MoveFirst                        ' Перейти на первую запись
                        Do While Not .EOF                  ' Поиск нужной детали в таблице Детали
                            lCodePart = .Fields("КодДетали") ' Код детали в табл. Детали
                            If lCodePart = lCodePartShip Then ' Если данная деталь поставлялась,
                                strParts = strParts + .Fields("Название") + ", " ' добавить ее назв.
                                Exit Do                    ' Досрочный выход из цикла
                            End If
                        .MoveNext                          ' Переход к следующей детали таблицы Детали
                    Loop                                  ' Конец цикла для таблицы Детали
                End With
            End If
            .MoveNext                                  ' Переход к следующей записи таблицы Поставки
        Wend                                            ' Конец цикла по таблице Поставки
        If strParts <> "" Then                          ' Если поставщик что-то поставлял
            With rstDeliveredParts                      ' Для таблицы ПоставленныеДетали
                .AddNew                                  ' Добавляем новую запись
                .Fields("Поставщик") = strSupplier      ' Заполняем поле Поставщик
                ' Убираем последнюю запятую из списка деталей
            End With
        End If
    End While
End With

```

```

        strParts = Mid(strParts, 1, InStrRev(strParts, ",") - 1)
        .Fields("Детали") = strParts ' Заполняем поле Детали
        .Update                      ' Сохранение добавленной записи в таблице
    End With
End If
End With
.MoveNext                          ' Переход к следующей записи таблицы Поставщики
Wend                               ' Конец цикла по таблице Поставщики
End With
rstSuppliers.Close                 ' Заккрытие таблиц
rstParts.Close
rstShipments.Close
rstDeliveredParts.Close
End Sub

```

Для работы с базой данных в программе следует иметь объект типа *Database*, которому должно быть присвоено значение некоторой базы данных. Метод *CurrentDb* возвращает объект типа *Database*, соответствующий базе данных, открытой в данный момент в окне Access. В программе переменной *db* присваивается значение, возвращаемое *CurrentDb*.

Доступ к таблицам базы данных в программах можно осуществить несколькими способами. В данной программе созданы четыре переменные типа *DAO.RecordSet*, которые могут представлять наборы данных из таблиц или запросов. Объекты *RecordSet* определены в нескольких библиотеках, в данном случае использована библиотека *DAO*.

Наборы данных создаются методом *OpenRecordset* объекта *Database*. В качестве аргумента этому методу передаются имена таблиц БД.

В программе использованы следующие методы объекта *RecordSet*:

- *MoveFirst* – переход на первую запись;
- *MoveNext* – переход к следующей записи;
- *Delete* – удаление текущей записи;
- *Update* – сохранение изменений в наборе данных.

Использованы также следующие свойства объекта *RecordSet*:

- *EOF* – имеет значение *True* (истина), если текущая позиция в таблице находится после последней записи;
- *Fields* – коллекция всех полей объекта *RecordSet*, доступ к отдельному полю производится по имени поля.

Для удаления последней запятой из списка названий деталей использованы две функции, работающие со строками:

InStrRev(s1, s2) – возвращает номер символа строки *s1*, начиная с которого в *s1* входит *s2*, причем поиск идет с конца строки *s1*, например, выражение *InStrRev(strParts, ",")* дает номер последней запятой в *strParts*;

Mid(s, start, length) – возвращает подстроку строки *s* длиной *length* символов, начиная с символа *start*.

Содержимое таблицы *ПоставленныеДетали*, сформированное программой приведено на рис. 2.18.

ПоставленныеДетали : таблица		
	Поставщик	Детали
▶	Прииск	Гайка, Болт, Винт, Винт, Муфта, Шпилька
	Макрос	Гайка, Болт
	Виктория	Болт
	Астра	Болт, Винт, Муфта, Гайка
*		
Запись: 1 из 4		

Рис. 2.18. Таблица, заполненная программой

Созданный с помощью мастера отчет для этой таблицы показан на рис. 2.19. Таким образом, поставленная задача решена.

ПоставленныеДетали	
<i>ПоставленныеДетали</i>	
<i>Поставщик</i>	<i>Детали</i>
Прииск	Гайка, Болт, Винт, Винт, Муфта, Шпилька
Макрос	Гайка, Болт
Виктория	Болт
Астра	Болт, Винт, Муфта, Гайка
Страница: 1	

Рис. 2.19. Отчет по таблице *ПоставленныеДетали*

3. Элементы теории баз данных

Любая база данных хранит информацию о некоторых предметах или явлениях, связанных между собой различными способами. Объекты и связи между ними не зависят от применяемой СУБД, поэтому необходимо независимое средство для описания данных об объектах и о связях между объектами. Необходимость в удобных средствах моделирования становится особенно острой, когда моделируемая область сложна и содержит много объектов, данные о которых должны храниться в базе. Наибольшее распространение из нескольких методов описания предметной области баз данных получил метод «сущность-связь» Чена, предложенный в 1979 г.¹ Другое название этого метода – метод ER-диаграмм. Здесь Е происходит от Entity – сущность, R – от Relationship – связь. Нотация данного метода используется в автоматизированных системах проектирования информационных систем, называемых CASE-системами (Computer-Aided Software Engineering – компьютерная поддержка разработки программ).

3.1. Метод «сущность-связь»

3.1.1. Сущности и их атрибуты

Сущность – это нечто, о чем хранится информация. Сущностью может быть материальный объект, или, например, событие (концерт, обращение к врачу и т.п.).

Сущности описываются данными, называемыми *атрибутами* (attributes). Каждая группа атрибутов, описывающая одно реальное проявление сущности, представляет собой *экземпляр* (instance) сущности.

Атрибут или набор атрибутов, который позволяет отличить данный экземпляр сущности от любого другого экземпляра, называется *ключом сущности*. При сохранении в базе данных экземпляра сущности нужно, чтобы СУБД обеспечила наличие у нового экземпляра уникального ключа. Это является примером *ограничения* (constraint), то есть правила, которому должны удовлетворять данные.

3.1.2. Домены

Множество значений, которые может принимать атрибут сущности, называется *доменом*. Домен может совпадать с типом данных, назначаемым полю таблицы базы данных, например, доменом может быть множество значений целого типа. При проектировании базы данных можно назначить в качестве домена перечень возможных значений атрибута, например, для цвета детали можно назначить 7 названий цветов. Атрибут может принимать текстовые значения, которые всегда ограничиваются определенной длиной строки, тогда домен будет представлять множество строк текста, длина которых не превышает заданного значения. Множество возможных элементов домена можно ограничить также маской ввода.

3.1.3. Связи между сущностями

Пусть имеются две сущности *A* и *B*. Говорят, что между этими сущностями установлена связь «один-к-одному» (one-to-one relationship), если любому экземпляру *A_i* сущности *A* соответствует единственный экземпляр сущности *B* или не соответствует ни од-

¹ Chen P.P.S. The Entity-Relationship Model – Toward a Unified View of Data//ACM Transactions on Database Systems.–1979, vol.1, no. 1, pp.9-36.

ного, а экземпляру B_j сущности B соответствует один экземпляр сущности A или ни одного.

Пусть сущность A это небольшие города, а сущность B – аэропорты. Тогда между A и B можно установить связь «один-к-одному», так как в небольших городах может быть только один аэропорт. Другим примером связи «один-к-одному» является брак между мужчиной и женщиной.

Связь «один-к-одному» встречается достаточно редко, так как часто при внимательном анализе оказывается, что там, где вначале представлялись две сущности, имеющие связь вида «один-к-одному» в действительности имеется *одна* сущность, объединяющая атрибуты, ранее относимые к двум сущностям.

Самым распространенным видом связей является связь «один-ко-многим» (one-to-many relationship). Между сущностями A и B имеется связь «один-ко-многим», если одному экземпляру A_i сущности A соответствует нуль, один или более экземпляров сущности B , а экземпляру B_j сущности B соответствует нуль или *один* экземпляр сущности A . Например, у женщины может не быть детей, может быть один ребенок или несколько детей, то есть между сущностью *Женщины* и сущностью *Дети* имеется связь «один-ко-многим».

Связь «многие-ко-многим» (many-to-many relationship) между двумя сущностями A и B имеет место, если одному экземпляру A_i сущности A соответствует нуль, один или более экземпляров сущности B , а экземпляру B_j сущности B соответствует нуль, один или более экземпляров сущности A . Например, на факультете можно выделить сущность *Преподаватель* и сущность *Предмет*. Между этими сущностями имеется связь «многие-ко-многим», так как один преподаватель может читать несколько предметов, а один предмет может читаться несколькими преподавателями (в разных группах или потоках). Связь «многие-ко-многим» не поддерживается непосредственно реляционной моделью данных и сводится к связям «один-ко-многим» путем ввода промежуточной сущности. Для сущностей *Преподаватель* и *Предмет* такой промежуточной сущностью может быть сущность *Расписание*, где можно хранить информацию о преподавателях, читаемых ими предметах и потоках или группах, которые слушают предмет.

3.1.4. Слабые сущности и обязательные связи

Рассмотрим две сущности: *Женщины* и *Дети*, между которыми имеется связь вида «один-ко-многим». Какая-либо женщина может не иметь детей, но ребенок всегда имеет мать, то есть экземпляр сущности *Дети* **обязательно** связан с одним экземпляром сущности *Женщины*. Сущность, которая **обязательно** должна быть связана с другой сущностью, называется *слабой*. В данном примере слабой сущностью является сущность *Дети*.

3.1.5. Документирование сущностей и связей

Сущности обозначаются прямоугольниками, а связи – линиями. На концах линий указывают характеристики связи, например, ER-диаграмму для БД *ПоставщикиДетали* можно изобразить в виде, показанном на рис.3.1.



Рис. 3.1. Диаграмма «сущность-связь» для БД *ПоставщикиДетали*

Значки, указываемые на концах линий связи, обозначают следующее:

- | – один, обязательная связь;
- | 0 – нуль или один, необязательная связь;
- < – один или много, обязательная связь;
- 0< – нуль, один или более, необязательная связь.

3.2. Реляционная модель данных

Реляционная модель данных основана на теории множеств. Она предложена Эдгаром Коддом в 1970 г.² Название модели происходит от слова relation – *отношение*.

3.2.1. Отношения

Пусть есть n множеств D_1, D_2, \dots, D_n . Отношением R над этими множествами называется множество n – *кортежей* вида $\langle d_1, d_2, \dots, d_n \rangle$, где $d_i \in D_i, i = 1, 2, \dots, n$. Множества D_i называются *доменами*. Кортеж – это множество из n элементов, причем каждый из этих элементов берется из своего домена.

Значение, которое берется из домена, является значением *атрибута* отношения. Отношения создаются, чтобы хранить данные о каком-либо предмете или явлении (о сущности), поэтому понятие *атрибута отношения* совпадает с понятием *атрибута сущности*, то есть атрибут – это некоторое свойство объекта, который моделируется сущностью или отношением.

Значение атрибута всегда *одинокое*, не допускается одновременно несколько значений для атрибута.

Первичным ключом отношения называется атрибут или набор атрибутов с уникальными значениями. Первичный ключ позволяет отличать один кортеж от другого.

Количество n атрибутов в отношении называется *степенью* отношения.

Количество кортежей в отношении называется *мощностью* отношения.

Для отношений используется следующее обозначение:

ИмяОтношения(ПервичныйКлюч, Атрибут1, ...)

Например, отношение *Детали*:

Детали(КодДетали, Название, Вес, Цвет, Эскиз)

Отношения имеют наглядное представление в виде двумерных таблиц, например, для отношения *Поставщики* эта таблица может иметь вид:

<i>Поставщики</i>			
<u>КодПоставщика</u>	Поставщик	Город	Рейтинг
1	Прииск	Н.Новгород	100
2	Макрос	Н.Новгород	100
3	Виктория	Кстово	60
4	Астра	Москва	90
5	Глория	Казань	25
6	Италмаз	Казань	25

Таблица имеет *заголовок*, в котором размещаются названия атрибутов. Строками таблицы являются кортежи. Приведенное отношение *Поставщики* имеет 6 кортежей. Строки таблицы образуют *тело* отношения.

² Codd E.F. A Relational Model of Data for Large Shared Data Banks//Communications of the ACM.– 1970, Vol.13, No.6

Моделирование отношений в виде таблиц не является полностью адекватным реляционной модели, в которой используется понятие множества. Например, таблица может иметь одинаковые строки, а отношение, как множество, не допускает одинаковых кортежей. Далее, кортеж есть *множество* атрибутов, поэтому безразличен порядок следования атрибутов в кортеже, столбцы же таблицы, вообще говоря, упорядочены.

Список имен атрибутов отношения называется *схемой отношения*.

*Экземпляр*ом отношения является его значение в некоторый момент времени.

Схемы двух отношений называются *эквивалентными*, если они имеют одинаковую степень и возможно такое их упорядочение, что на одном месте будут стоять атрибуты с одинаковыми доменами.

3.2.2. Виды отношений

Реляционная база данных состоит из набора отношений.

Отношение, непосредственно хранящееся в базе данных, называется *базовым*.

Отношения, получаемые путем выполнения операций над базовыми отношениями, называются *представлениями*.

3.2.3. Сущности и отношения

При проектировании базы данных преследуются следующие цели:

- В базе данных должна храниться вся необходимая информация о предметной области;
- В базе данных не должно быть избыточной информации.

Метод «сущность-связь» позволяет достичь целей проектирования, выявляя объекты предметной области, о которых должна храниться информация в базе, и связи между объектами. При переходе к реализации разработанной модели каждая сущность представляется одним отношением.

3.3. Реляционная алгебра

Над отношениями можно выполнять 8 операций, в результате которых получаются новые отношения. Это 4 традиционные теоретико-множественные операции: *объединение*, *пересечение*, *вычитание* и *декартово произведение* и 4 специальные реляционные операции: *выборка*, *проекция*, *соединение* и *деление*.

С точки зрения теории множеств, операции объединения, пересечения, вычитания применимы к любым множествам, то есть чисто формально можно объединить отношение *Поставщики* и отношение *Детали*. Но отношения – это множества, элементами которого являются кортежи, а при формальном объединении различных отношений получится неоднородная смесь кортежей, не являющаяся отношением, поэтому для допустимости операций объединения, пересечения, вычитания требуется, чтобы схемы отношений были эквивалентными, то есть, чтобы отношения были совместимы по типу. Данное ограничение обеспечивает *замкнутость* операций реляционной алгебры, то есть результат операции над отношениями также является отношением.

Объединением двух отношений A и B ($A \cup B$) называется отношение, состоящее из множества всех кортежей, принадлежащих A или B или обоим отношениям.

Пересечением двух отношений A и B ($A \cap B$) называется отношение, состоящее из множества всех кортежей, принадлежащих одновременно A и B .

Вычитанием двух отношений A и B ($A - B$) называется отношение, состоящее из множества всех кортежей, принадлежащих A и не принадлежащих B .

Декартовым произведением двух отношений A и B ($A \times B$) называется отношение, состоящее из множества всех кортежей, полученных сцеплением каждого кортежа

отношения A с каждым кортежем отношения B . Под сцеплением двух кортежей $\langle A1, A2, \dots, Am \rangle$ и $\langle B1, B2, \dots, Bn \rangle$ понимается кортеж, полученный объединением всех атрибутов в один кортеж $\langle A1, A2, \dots, Am, B1, B2, \dots, Bn \rangle$.

Выборка позволяет отобрать из отношения только часть кортежей, удовлетворяющих некоторому условию. Пусть X и Y – атрибуты отношения A , а θ оператор сравнения ($>$, $<$, $=$ и т.д.).

Выборкой

(A WHERE $X \theta Y$)

называется отношение, содержащее кортежи отношения A , для которых условие $X \theta Y$ является истинным. В СУБД выборка реализуется в виде фильтра или запроса, которые выделяют часть записей таблицы.

Проекцией отношения A по атрибутам X, Y, \dots, Z , где каждый из атрибутов принадлежит отношению A , называется отношение с заголовком $\{X, Y, \dots, Z\}$ и телом, содержащим кортежи, получаемые из кортежей A после вычеркивания из них атрибутов, не принадлежащих $\{X, Y, \dots, Z\}$. Например, проектирование отношения *Поставщики* на атрибуты $\{\text{Город}, \text{Рейтинг}\}$ даст отношение:

Поставщики[Город, Рейтинг]

Город	Рейтинг
Н.Новгород	100
Кстово	60
Москва	90
Казань	25

При проекции удаляются столбцы таблицы и исключаются повторяющиеся строки.

Пусть имеется отношение $A(X, Y)$ и отношение $B(Y, Z)$. **Естественным соединением** отношений A и B (A JOIN B) называется отношение $C(X, Y, Z)$ с кортежами $\langle X, Y, Z \rangle$ такими, что $\langle X, Y \rangle$ входит в отношение A , а $\langle Y, Z \rangle$ входит в B . На рис.3.2 приведен конкретный пример соединения двух отношений.

$A(X, Y)$		$B(Y, Z)$		$C(X, Y, Z) = A \text{ JOIN } B$		
X	Y	Y	Z	X	Y	Z
1	10	20	A	2	60	B
2	60	60	B	3	90	D
3	90	70	C			
4	50	90	D			
5	25					

Рис. 3.2. Пример естественного соединения двух отношений

Понятие естественного соединения распространяется на случай, когда X, Y, Z являются не одиночными атрибутами, а множествами атрибутов.

Пусть имеется отношение $A(X, Y)$ и отношение $B(Y)$. **Делением** отношений A и B (A DIVIDE B) называется отношение $C(X)$, кортежи которого $\langle X \rangle$ входят в отношение A и которым соответствуют кортежи $\langle Y \rangle$, входящие в отношение B . Ниже приведен пример.

$A(X, Y)$		$B(Y)$	$C(X) = A \text{ DIVIDE } B$
X	Y	Y	X
1	A	A	1
2	B	C	3
3	C	E	5
4	D	F	
5	E		

Рис. 3.3. Пример выполнения деления отношений

По аналогии с единичными атрибутами X, Y, Z понятие деления распространяется на случай, когда X, Y, Z являются множествами атрибутов.

Операции над отношениями реализованы в виде команд языка SQL. С помощью этих команд можно делать *выборку* данных, производить *обновление* данных, формулировать *правила безопасности* для контроля доступа к данным, определять *правила целостности*.

3.4. Нормализация отношений

3.4.1. Цели проектирования базы данных

База данных должна адекватно отображать предметную область, быть эффективной, содержать актуальную информацию. Более конкретно цели проектирования можно сформулировать в виде:

- База данных должна хранить всю необходимую информацию, для чего следует учесть все атрибуты и распределить их по отношениям;
- Исключение дублирования данных;
- Сведение числа хранимых отношений к минимуму.

Заметим, что вторая и третья цели противоречат друг другу, так как для исключения дублирования данных одно отношение разбивается на два или более.

Для того чтобы отношения удовлетворяли целям проектирования, они должны отвечать определенным требованиям, которые называются *нормальными формами*. Всего существует шесть нормальных форм, их схема показана на рис.3.4. Если отношение удовлетворяет высшей нормальной форме, то оно удовлетворяет и нормальной форме с меньшим номером.



Рис. 3.4. Нормальные формы

3.4.2. Первая нормальная форма (1НФ)

Атомарным называется неделимое значение из некоторого домена. Множество значений не является атомарным значением. Например, в библиографических карточках библиотечного каталога указаны автор книги, ее название и список инвентарных номеров экземпляров данной книги, имеющихся в библиотеке. Здесь при моделировании в виде отношения библиографических карточек атрибут *ИнвентарныйНомер* мог бы иметь множественное значение, то есть данный атрибут не является атомарным.

Говорят, что отношение находится в *первой нормальной форме*, если каждый его атрибут имеет, и всегда будет иметь атомарное значение.

В примере с библиографическими карточками есть еще один потенциально многозначный атрибут – автор, так как существуют книги, написанные в соавторстве несколькими писателями. Таким образом, отношение

Карточка(Автор, Название, МестоИздания, Издательство, Год, КолСтраниц, ИнвНомер) не находится в 1НФ.

Для приведения отношения к 1НФ следует устранить многозначность для чего нужно разбить отношение на два или более отношений, и в отдельные отношения включить многозначные атрибуты. Вместо одного отношения *Карточка* создадим четыре отношения:

Автор(КодАвтора, Фамилия, Имя, Отчество, ДатаРождения)

Книга(КодКниги, Название, МестоИзд, Издательство, Год, КолСтраниц)

Авторство(КодКниги, КодАвтора)

Наличие(КодКниги, ИнвНомер)

Данные отношения находятся уже в 1НФ.

Первая нормальная форма имеет ряд недостатков. Вернемся к задаче о поставщиках и деталях. Учет поставок можно вести с помощью одного отношения *Накладные*, содержащего всю необходимую информацию:

Накладные(НазвПоставщика, Город, Рейтинг, НазвДетали, ЦветДет, ВесДет, ЭскизДет, Количество, ДатаПост).

Данное отношение находится в 1НФ, но имеются следующие недостатки:

- Избыточность информации, так придется повторно вносить данные о каждом поставщике при каждой поставке им детали и данные о детали, поставленной еще раз;
- Невозможно ввести в базу данные о поставщике, если он еще не осуществил какую-либо поставку и данные о детали, которая ни разу не поставлялась;
- При изменении, например, адреса поставщика, придется просматривать все corteжи и вносить в них изменения;
- При удалении единственного corteжа, в котором есть сведения о поставщике, полностью будет потеряна информация о данном поставщике (аналогично – о детали).

3.4.3. Функциональные зависимости

Понятие функциональной зависимости позволяет обеспечить более рациональное проектирование структуры отношений базы данных.

Пусть каждому значению атрибута X отношения R соответствует ровно одно, связанное с ним значение атрибута Y , тогда говорят, что Y функционально зависит от X . Функциональная зависимость обозначается $X \rightarrow Y$.

Возможным ключом отношения называется атрибут или набор атрибутов, который может быть использован для данного отношения в качестве первичного ключа.

Рассмотрим базу данных для учета успеваемости по итогам некоторого семестра, включающую три отношения:

Студенты(НомСтуд, ФИОСтуд),

Предметы(КодПредм, НазвПредм),

Оценки(КодПредм, НомСтуд, Балл, Группа, Спец).

В отношении *Студенты* имеется, например, функциональная зависимость:

НомСтуд -> *ФИОСтуд*.

В отношении *Оценки* первичным ключом является комбинация {*КодПредм*, *НомСтуд*} и имеются функциональные зависимости, показанные на рис.3.5.

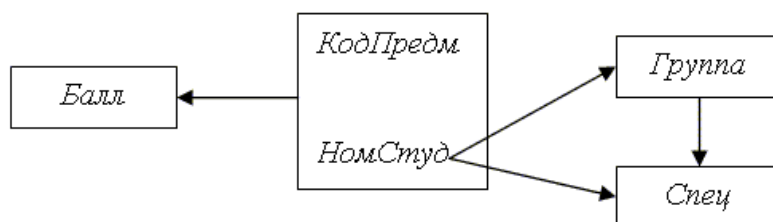


Рис. 3.5. Функциональные зависимости для отношения *Оценки*

3.4.4. Вторая нормальная форма

Говорят, что отношение находится во *второй нормальной форме* (2НФ), если его неключевые атрибуты полностью зависят от первичного ключа.

Для отношения *Оценки* это не так, потому что атрибуты *Группа* и *Спец* зависят только от одной части первичного ключа – атрибута *НомСтуд*. Разобьем отношение *Оценки* на два:

Успеваемость(КодПредм, НомСтуд, Балл) и

ГрСпец(НомСтуд, Группа, Спец).

Диаграмма функциональных зависимостей для этих отношений представлена на рис.3.6.

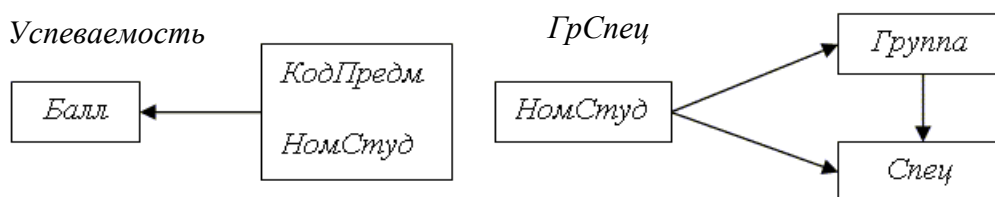


Рис. 3.6. Диаграмма ФЗ для отношений *Успеваемость* и *ГрСпец*

Данные отношения находятся во 2НФ.

3.4.5. Третья нормальная форма

Отношение находится в *третьей нормальной форме*, если оно находится во 2НФ, неключевые атрибуты взаимно независимы, и каждый неключевой атрибут неприводимо зависит от первичного ключа (то есть можно изменять значения атрибутов без изменения первичного ключа и других неключевых атрибутов).

Отношение *ГрСпец* не находится в 3НФ, так как в нем есть функциональная зависимость между неключевыми атрибутами:

Группа -> *Спец*.

Разобьем отношение *ГрСпец* на два:

СтудГр(НомСтуд, *Группа*) и
СтудСпец(НомСтуд, *Спец*).

Эти два отношения находятся в 3НФ.

Обычно бывает достаточно довести процесс проектирования отношений до третьей нормальной формы, при этом большинство проблем бывает устранено.

В нашем примере проектирование базы данных для учета текущей успеваемости студентов можно остановить, когда будет выяснено, что должны быть созданы следующие отношения:

Студенты(НомСтуд, *ФИОСтуд*),
Предметы(КодПредм, *НазвПредм*),
Успеваемость(КодПредм, НомСтуд, *Балл*),
СтудГр(НомСтуд, *Группа*),
СтудСпец(НомСтуд, *Спец*).

4. Язык SQL

Язык структурированных запросов Structured Query Language (аббревиатура SQL произносится как «эс-кью-эл», иногда «си-квэл») является стандартным языком обработки данных, используемым большинством СУБД.

4.1. Краткая история SQL

Этот язык был предложен исследовательской лабораторией фирмы IBM в начале 1970-х годов для реализации реляционной модели данных Э.Ф.Кодда (E.F.Codd).

Существует ряд стандартов языка SQL, разработанных Американским национальным институтом стандартов (ANSI – American National Standards Institute), на базе которых приняты международные стандарты ISO (International Standards Organization).

В 1989г. принят стандарт SQL-89 (или SQL1). Его реализуют большинство коммерческих СУБД.

Следующий стандарт SQL-92 (или SQL2) был принят в 1992 г. В нем произведена существенная редакция первоначального стандарта. Язык SQL очень объемный, его полное описание в стандарте занимает более 600 с. Не все возможности, предусмотренные в стандарте, пока реализуются имеющимися СУБД.

В 1999 г. появился новый стандарт SQL-99 (или SQL3), в который включен ряд концепций из объектно-ориентированного программирования.

Язык SQL был разработан задолго до появления графических интерфейсов пользователя, поэтому он ориентирован на использование текста для написания команд. Рассмотренная выше работа с базами данных в среде Access выполнялась исключительно средствами графического интерфейса. Но, как правило, это стандартные типовые действия, которые предусмотрели разработчики интерфейса. Использование SQL позволяет выполнять более гибкую обработку данных, например, путем встраивания команд SQL в текст программ на каком-либо языке программирования.

С помощью SQL можно определять структуры данных, а также запрашивать и обновлять информацию в базе данных. Совокупность команд, служащих для определения данных, называют *языком определения данных* (Data Description Language, DDL), а совокупность команд для обновления и запроса данных – *языком манипулирования данными* (Data Manipulation Language, DML).

4.2. SQL в Access

Многие СУБД поддерживают интерактивный режим работы с интерфейсом в виде командной строки, в которой можно вводить команды языка SQL и выполнять их по нажатию Enter. Такого режима в Access нет, но возможность вводить и выполнять команды SQL имеется, например, в SQL-окне конструктора запросов. Команды SQL могут также быть частью программного кода на VBA.

Пусть открыта база данных *ПоставщикиДетали*. Перейдем в раздел Запросы окна базы данных и нажмем кнопку Создать на панели инструментов.

В диалоговом окне Новый запрос, рис. **Ошибка! Источник ссылки не найден.**, выберем в списке строку Конструктор и нажмем кнопку ОК, откроется окно конструктора запросов и окно **Добавление таблицы**. Не добавляя таблицы или запроса, нажмем в диалоговом окне **Добавление таблицы** кнопку **Заккрыть**. На экране останется пустой бланк запросов.

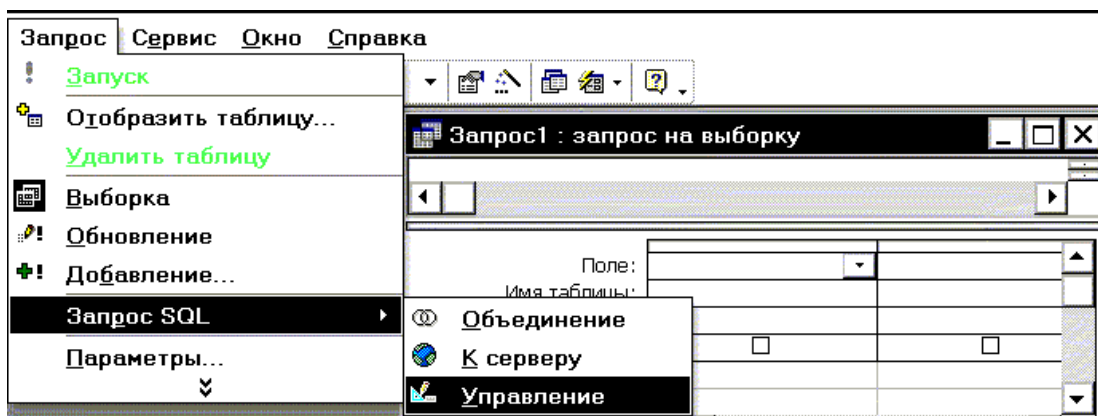


Рис. 4.1. Переход к созданию SQL запроса

В меню **Запрос** выберем команду **Запрос SQL** и подкоманду **Управление**, рис.4.1. На экране появится окно для ввода инструкций языка SQL.

Введем инструкцию SQL для управляющего запроса, рис.4.2, которая отберет названия всех поставщиков из таблицы *Поставщики*. После нажатия кнопки данный запрос будет выполнен, и мы увидим названия всех поставщиков из таблицы *Поставщики*, рис.4.3.

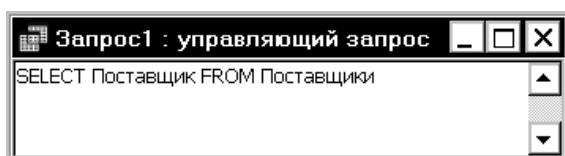


Рис. 4.2. Ввод управляющего запроса

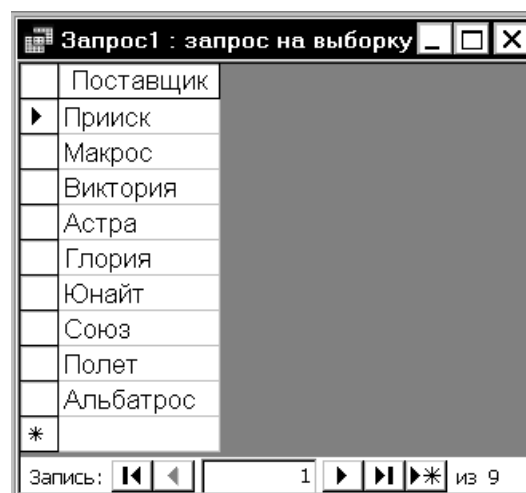


Рис. 4.3. Результат выполнения управляющего запроса

Переключаться между просмотром результатов запроса и режимом SQL можно с помощью кнопки **Вид** на панели инструментов или с помощью команд меню **Вид**, рис.4.4.

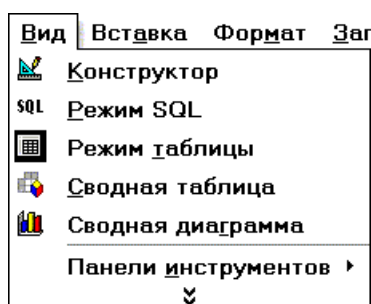


Рис. 4.4. Меню **Вид** для запросов

Каждый управляющий запрос может содержать только одну управляющую инструкцию.

Справочная система Access содержит описание используемой версии языка SQL. Для получения доступа к ней надо выполнить команду **Справка, Справка: Microsoft Access**. Далее в окне справки нужно перейти на вкладку **Содержание** и выбрать нужный раздел, рис. 4.5.

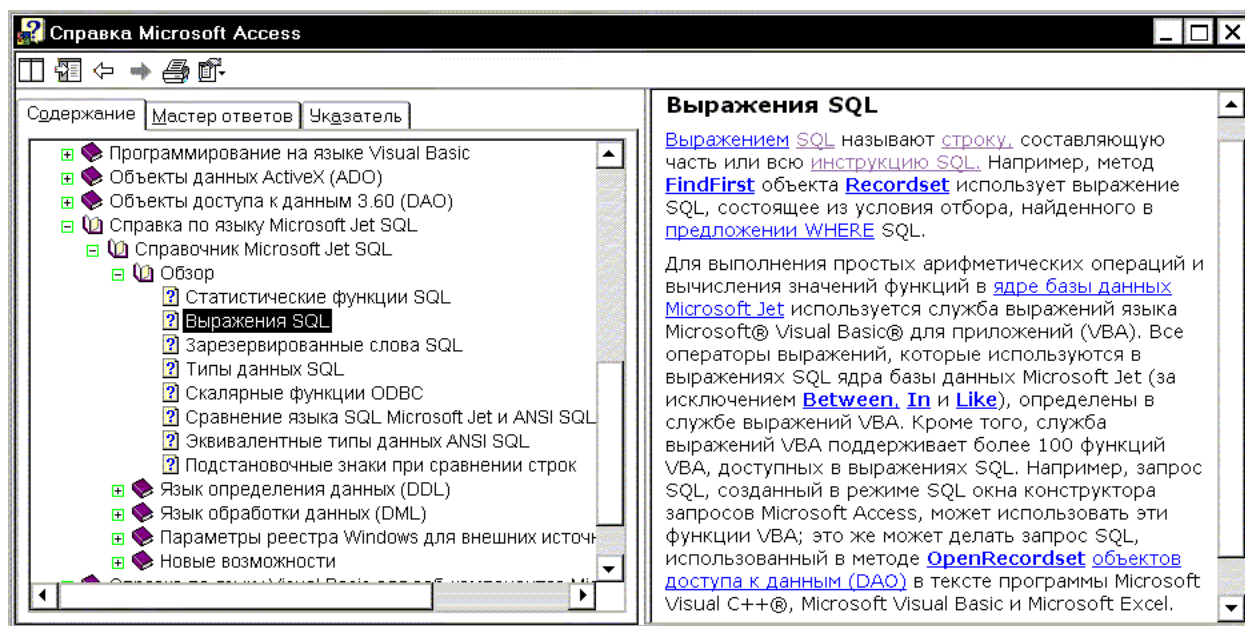


Рис. 4.5. Справка по языку SQL

Название Microsoft Jet имеет ядро базы данных, то есть та часть системы управления базами данных, которая отвечает за загрузку и сохранение данных в пользовательских и системных базах данных. Ядро базы данных Microsoft Jet можно рассматривать как диспетчер данных, на основе которого строятся другие системы доступа к данным, такие как Microsoft Access и Visual Basic. Таким образом, Access является надстройкой над ядром базы данных Microsoft Jet.

Как видно из рис. 4.5, в системе справки приводится полный перечень используемых ключевых слов SQL, дается сравнение версии Microsoft Jet SQL со стандартом ANSI, в котором отмечается ее соответствие, в основном, стандарту SQL-89.

При описании языка используются термины *инструкция* и *предложение*. Инструкция – это полностью законченная команда языка SQL. Инструкции заканчиваются точкой с запятой. Предложение – это часть инструкции, имеющая самостоятельное значение.

4.3. Работа с таблицами

Создание таблиц, индексов и других объектов можно осуществлять с помощью удобного конструктора, однако есть возможность использовать для этого управляющие запросы SQL. Управляющие запросы SQL используются для создания, удаления или изменения таблиц или для создания индексов в текущей базе данных.

Microsoft Access поддерживает следующие управляющие инструкции SQL:

CREATE TABLE — создает таблицу;

ALTER TABLE — добавляет новое поле или ограничение в существующую таблицу;

DROP — удаляет таблицу из базы данных или удаляет индекс, определенный для поля или группы полей;

CREATE INDEX — создает индекс для поля или группы полей.

4.3.1. Создание таблиц

Новая таблица создается инструкцией CREATE TABLE.

Синтаксис инструкции:

```
CREATE [TEMPORARY] TABLE таблица  
(поле_1 тип [(размер)] [NOT NULL] [WITH COMPRESSION | WITH COMP] [индекс_1]  
[, поле_2 тип [(размер)] [NOT NULL] [индекс_2] [, ...]]  
[, CONSTRAINT составнойИндекс [, ...]])
```

Ключевые слова при описании синтаксиса записываются заглавными буквами, обязательные элементы заключаются в квадратные скобки, вертикальная черта разделяет различные варианты написания предложений.

Описание аргументов инструкции CREATE TABLE: приведено в Табл. 4.1.

Таблица 4.1. Аргументы инструкции CREATE TABLE

Элемент	Описание
таблица	Имя создаваемой таблицы.
поле_1, поле_2	Имена одного или нескольких полей, создаваемых в новой таблице. Таблица должна содержать хотя бы одно поле.
тип	Тип данных поля в новой таблице.
размер	Размер поля в знаках (только для текстовых и двоичных полей).
индекс_1, индекс_2	Предложение CONSTRAINT, предназначенное для создания простого индекса.
составнойИндекс	Предложение CONSTRAINT, предназначенное для создания составного индекса.

Инструкция CREATE TABLE используется для описания новой таблицы, ее полей и индексов. Если для поля добавлено ограничение NOT NULL, то при добавлении новых записей это поле должно содержать непустое значение.

Предложение CONSTRAINT устанавливает различные ограничения на поле и может быть использовано для определения ключа. Кроме того, для создания ключа или дополнительного индекса для существующей таблицы можно использовать инструкцию CREATE INDEX.

Допускается использование ограничения NOT NULL для одиночного поля, а также внутри именованного предложения CONSTRAINT, которое применяется к одиночному полю или к именованному предложению CONSTRAINT, предназначенному для создания составного индекса. Однако ограничение NOT NULL можно наложить на поле только один раз. При попытке применить это ограничение несколько раз возникает ошибка выполнения.

Создаваемая временная (TEMPORARY) таблица будет доступна только в том сеансе, где эта таблица была создана. По завершении данного сеанса она автоматически удаляется. Временные таблицы могут быть доступны для нескольких пользователей.

Использование атрибута WITH COMPRESSION допускается только для типов данных CHARACTER и MEMO (он же TEXT) и их синонимов.

Атрибут WITH COMPRESSION был добавлен к столбцам CHARACTER вследствие перехода к формату представления знаков Юникод. Каждый знак в формате Юникод всегда кодируется с помощью двух байтов. Для существующих баз данных Microsoft Jet, содержащих в основном символьные данные, это может означать увеличение размера файла базы данных примерно в два раза после преобразования в формат Microsoft Jet версии 4.0. Тем не менее, для многих наборов символов, ранее обозначавшихся как однобайтовые наборы символов (SBCS), представление в формате Unicode (SBCS) может быть без труда сжато до одного байта. Если столбец CHARACTER был определен с этим атрибутом, то при сохранении в нем данных осуществляется их автоматическое сжатие, а при извлечении данных – обратная операция.

Столбцы MEMO также могут быть определены для хранения данных в сжатом формате. Однако при этом действует одно ограничение. Сжатию подвергаются только те столбцы типа MEMO, которые в сжатом виде имеют размер не более 4096 байтов. Все остальные столбцы MEMO не сжимаются. Это означает, что для данной таблицы и данного столбца MEMO этой таблицы одни данные могут быть сжаты, а другие – нет.

В следующем управляющем запросе с помощью инструкции CREATE TABLE создается таблица *Друзья*. Приведенная инструкция определяет имена и типы полей таблицы и создает для поля *Код* индекс, назначающий это поле ключевым.

```
CREATE TEMPORARY TABLE Друзья
([Код] integer,
[Фамилия] text,
[Имя] text,
[ДеньРождения] date,
[Телефон] text (10),
[Примечания] memo,
CONSTRAINT [Индекс1] PRIMARY KEY ([Код]));
```

Для выполнения данной инструкции нужно создать отдельный управляющий запрос, как это описано выше в п.11.2. Созданная таблица в режиме конструктора показана на рис. 4.6. Свойству **Индексированное поле** поля *Код* установлено значение **Да (Совпадение не допускаются)**. Размер текстовых полей инструкцией SQL установлен максимально возможный – 256.

Имя поля	Тип данных	Описание
Код	Числовой	
Фамилия	Текстовый	
Имя	Текстовый	
ДеньРождения	Дата/время	
Телефон	Текстовый	
Примечания	Поле MEMO	

Свойства поля	
Общие	
Размер поля	Длинное целое
Формат поля	
Число десятичных знаков	Авто
Маска ввода	
Подпись	
Значение по умолчанию	
Условие на значение	
Сообщение об ошибке	
Обязательное поле	Нет
Индексированное поле	Да (Совпадения не допускаются)

Имя поля может состоять из 64 знаков с учетом пробелов. Для справки по именам полей нажмите клавишу F1.

Рис. 4.6. Таблица *Друзья* в режиме конструктора

4.3.2. Создание индекса

При помощи индексов ускоряется сортировка и поиск записей. Индексы таблиц Microsoft Access используются так же, как предметные указатели в книгах: при поиске данных выполняется их поиск в индексе. Индексы можно создавать по одному или нескольким полям. Составные индексы позволяют пользователю различать записи, в которых первые поля могут иметь одинаковые значения.

В основном требуется индексировать поля, в которых часто осуществляется поиск. Однако индексы могут замедлить выполнение некоторых запросов на изменение, напри-

мер, запросов на добавление, при выполнении которых требуется обновление индексов многих полей.

Поля первичного ключа таблиц индексируются автоматически, а поля с типом данных **Поле объекта OLE** индексировать нельзя. Для остальных полей индексирование используется, если выполняются следующие условия.

Если предполагается частое выполнение одновременной сортировки или поиска в нескольких полях, можно создать для этих полей составной индекс. Например, если в одном и том же запросе часто задаются условия для полей *Имя* и *Фамилия*, то для этих двух полей имеет смысл создать составной индекс.

При сортировке таблицы по составному индексу Microsoft Access сначала выполняет сортировку по первому полю, определенному для данного индекса. Если в первом поле содержатся записи с повторяющимися значениями, то выполняется сортировка по второму полю, определенному для данного индекса, и так далее.

Следующая управляющая инструкция с помощью инструкции CREATE INDEX создает составной индекс по полям *Фамилия* и *Имя*.

```
CREATE INDEX NewIndex  
ON Друзья ([Фамилия], [Имя]);
```


Для просмотра созданных индексов откроем таблицу *Друзья* в режиме конструктора, рис. 4.6, и выполним команду **Вид, Индексы** или нажмем кнопку . Откроется окно с перечнем индексов и полей, по которым производится индексация, рис. 4.7.



Рис. 4.7. Окно индексов

Для создания индекса нужно ввести его имя в столбце **Индекс**, а в столбце **Имя поля** выбрать из списка поле таблицы, по которому будет производиться индексация.

Для удаления индекса нужно очистить содержимое строк, содержащих описание индекса.

4.4. Инструкция SELECT

По этой инструкции ядро базы данных Microsoft Jet возвращает данные из базы данных в виде набора записей.

Синтаксис инструкции SELECT имеет вид:

```
SELECT [предикат] { * | таблица.* | [таблица.]поле_1  
[AS псевдоним_1] [, [таблица.]поле_2 [AS псевдоним_2] [, ...]]}  
FROM выражение [, ...] [IN внешняяБазаДанных]  
[WHERE... ][GROUP BY... ]  
[HAVING... ]  
[ORDER BY... ]  
[WITH OWNERACCESS OPTION]
```

Аргументы инструкции SELECT перечислены в табл. 4.2.

Таблица 4.2. Аргументы инструкции SELECT

Элемент	Описание
<i>предикат</i>	Один из следующих предикатов отбора: ALL, DISTINCT, DISTINCTROW или TOP. Предикаты используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат ALL.
*	Указывает, что выбраны все поля заданной таблицы или таблиц.
<i>таблица</i>	Имя таблицы, из которой должны быть отобраны записи.
<i>поле_1, поле_2</i>	Имена полей, из которых должны быть отобраны данные. Если включить несколько полей, они будут извлекаться в указанном порядке.
<i>псевдоним_1, псевдоним_2</i>	Имена, которые станут заголовками столбцов вместо исходных названий столбцов в таблице.
<i>выражение</i>	Имена одной или нескольких таблиц, которые содержат отбираемые данные.
<i>внешняяБазаДанных</i>	Имя базы данных, которая содержит таблицы, указанные с помощью аргумента <i>выражение</i> , если они не находятся в текущей базе данных.

При выполнении этой операции ядро базы данных находит указанную таблицу или таблицы, извлекает заданные столбцы, выделяет строки, соответствующие условию отбора, и сортирует или группирует результирующие строки в указанном порядке.

Инструкции SELECT не изменяют данные в базе данных.

Обычно слово SELECT является первым словом инструкции SQL. Большая часть инструкций SQL является инструкциями SELECT или SELECT...INTO.

Ниже приведен минимальный синтаксис инструкции SELECT:

SELECT *поля* FROM *таблица*;

Для отбора всех полей таблицы можно использовать символ звездочки (*). Следующая инструкция отбирает все поля из таблицы *Сотрудники*:

SELECT * FROM *Сотрудники*;

Если несколько таблиц, включенных в предложение FROM, содержат одноименные поля, перед именем такого поля следует ввести имя таблицы и оператор (.) (точка). Предположим, что поле *Отдел* содержится в таблицах *Сотрудники* и *Начальники*. Следующая инструкция SQL отберет поле *Отдел* из таблицы *Сотрудники* и поле *Начальник* из таблицы *Начальники*:

```
SELECT Сотрудники.Отдел, Начальники.Начальник
FROM Сотрудники INNER JOIN Начальники
WHERE Сотрудники.Отдел = Начальники.Отдел;
```

4.4.1. Примеры использования инструкции SELECT

Рассмотрим несколько SQL-запросов к базе данных *ПоставщикиДетали*.
Запрос

SELECT Поставщик FROM Поставщики;

просто выведет наименования всех поставщиков из таблицы *Поставщики*.

Изменим предыдущий запрос и напишем:

```
SELECT Поставщик FROM Поставщики WHERE Город = "Москва";
```

Будут показаны поставщики из Москвы.

Создадим запрос:

```
SELECT Поставщик, Название FROM Поставщики, Детали WHERE Город="Москва";
```

В результате в запрос будут включены все поставщики из Москвы, и с каждым из них будут соединены все детали. Данный запрос реализует прямое декартово произведение таблиц и не имеет содержательного смысла.

Уточним запрос и покажем только детали, которые поставляют поставщики из Москвы:

```
SELECT Поставщик, Название, Количество
FROM Поставщики, Детали, Поставки
WHERE Город="Москва" And Поставщики.КодПоставщика=Поставки.КодПоставщика
AND Детали.КодДетали=Поставки.КодДетали;
```

В результате получим

Поставщик	Название	Количество
Астра	Болт	200
Астра	Винт	300
Астра	Муфта	400
Астра	Гайка	300

Рис. 4.8. Результат запроса на выборку из двух таблиц

Операция внутреннего соединения INNER JOIN объединяет записи из двух таблиц, если связующие поля этих таблиц содержат одинаковые значения. Она имеет синтаксис:

```
FROM таблица_1 INNER JOIN таблица_2
ON таблица_1.поле_1 оператор таблица_2.поле_2
```

Аргументы операции INNER JOIN перечислены в табл. 4.3.

Таблица 4.3. Аргументы операции INNER JOIN

Элемент	Описание
таблица_1, таблица_2	Имена таблиц, записи которых подлежат объединению.
поле_1, поле_2	Имена объединяемых полей. Если эти поля не являются числовыми, то должны иметь одинаковый тип данных и содержать данные одного рода, однако поля могут иметь разные имена.
оператор	Любой оператор сравнения: "=", "<", ">", "<=", ">=" или "<>".

Следующий запрос использует команду внутреннего соединения и выдает перечень всех поставщиков, код которых присутствует в таблице *Поставки*, причем каждый поставщик приводится столько раз, сколько поставок он сделал.

```
SELECT Поставщик FROM
Поставщики INNER JOIN Поставки
On Поставщики.КодПоставщика = Поставки.КодПоставщика;
```

Ограничиться только однократным упоминанием поставщика можно с помощью предиката DISTINCT:

```
SELECT DISTINCT Поставщик FROM
Поставщики INNER JOIN Поставки
```

ON Поставщики.КодПоставщика = Поставки.КодПоставщика;

Результат запроса показан на рис.4.9.

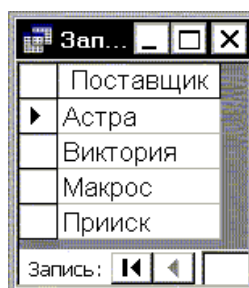


Рис. 4.9. Ограничение запроса предикатом DISTINCT

Сделаем внутреннее соединение таблицы *Поставщики* по полю *КодПоставщика* с таблицей, полученной в результате внутреннего соединения таблиц *Поставки* и *Детали* по полю *КодДетали*, и выберем поля *Поставщик*, *Название* и *Количество*:

```
SELECT Поставщик, Название, Количество
FROM Поставщики INNER JOIN
(Поставки INNER JOIN Детали ON Поставки.КодДетали = Детали.КодДетали)
ON Поставщики.КодПоставщика = Поставки.КодПоставщика;
```

Результат этого запроса показан на рис. 4.10.

Поставщик	Название	Количество
Прииск	Гайка	300
Прииск	Болт	200
Прииск	Винт	400
Прииск	Винт	200
Прииск	Муфта	100
Прииск	Шпилька	100
Макрос	Гайка	300
Макрос	Болт	400
Виктория	Болт	200
Астра	Болт	200
Астра	Винт	300
Астра	Муфта	400
Астра	Гайка	300
*		

Рис. 4.10. Результат запроса с двумя внутренними соединениями

С помощью предложения **WHERE** можно задать некоторое ограничивающее условие, например на дату поставки:

```
SELECT Поставщик, Название, Количество, Дата
FROM Поставщики INNER JOIN
(Поставки INNER JOIN Детали ON Поставки.КодДетали = Детали.КодДетали)
ON Поставщики.КодПоставщика = Поставки.КодПоставщика
WHERE Дата <= #6/30/2002#;
```

Дата здесь задается в американском формате: сначала месяц, затем день и год. Данный запрос показывает записи о поставках, которые произошли в первом полугодии 2002г., рис. 4.11.

Запрос1 : запрос на выборку				
	Поставщик	Название	Количество	Дата
▶	Прииск	Гайка	300	12.03.2002
	Прииск	Болт	200	06.05.2002
	Макрос	Гайка	300	14.04.2002
	Макрос	Болт	400	23.06.2002
*				
Запись: [Навигация] 1 [Навигация] из 4				

Рис. 4.11. Запрос с ограничением на дату

4.5. Использование SQL в VBA

В §2.9 приведено решение задачи об обработке данных с помощью программы на VBA, в которой использовались объекты типа DAO.Recordset связывались непосредственно с таблицами базы данных и использовались затем для выборки необходимых данных. Рассмотрим создание наборов данных, представляемых в программе объектами типа DAO.Recordset по результатам SQL запросов.

4.5.1. Классы для работы с запросами

Запросы к базе данных из программы на VBA можно выполнять с помощью объектов класса QueryDef. Запросы могут быть временными или постоянными. Для временных запросов не задается имя при их создании, а для постоянных запросов имя задается. Постоянные запросы включаются в коллекцию запросов QueryDefs, содержащую все запросы базы данных, к ним можно обращаться по имени.

Далее приведена программа, включающая одну функцию и одну процедуру.

В процедуре

CreateQueryDefX()

открывается база данных *ПоставщикиДетали*, создается временный запрос, в который отбираются все записи из таблицы *Поставщики* и создается также постоянный запрос, для отбора записей из таблицы *Детали*. Из этой процедуры вызывается функция

GetrstTemp(qdfTemp As QueryDef),

которая открывает запрос qdfTemp и печатает количество записей, отображенных в него.

Далее приводится текст программы.

' Пример программы, использующей класс QueryDef

' Функция GetrstTemp открывает набор записей по запросу qdfTemp и выводит отчет

' о количестве записей в наборе

Function GetrstTemp(qdfTemp As QueryDef)

Dim rstTemp As Recordset ' Переменная, представляющая набор записей

With qdfTemp

Debug.Print .Name ' Вывод в отладочное окно Immediate имени запроса

Debug.Print " " & .SQL

' Открытие набора записей из запроса.

' Константа dbOpenSnapshot устанавливает тип создаваемого набора записей:

' он не будет изменяться, если произойдут изменения в базе данных, пока не будет

' закрыт и снова не будет открыт

Set rstTemp = .OpenRecordset(dbOpenSnapshot)


```

With rstTemp
    ' Перемещение на последнюю запись набора и печать количества записей.
    .MoveLast
    Debug.Print " Number of records = " & _
        .RecordCount
    Debug.Print
    .Close
End With

End With

End Function

Sub CreateQueryDefX()
    Dim dbsSuppliersParts As Database      ' Переменная - база данных
    Dim qdfTemp As QueryDef               ' Временный запрос
    Dim qdfNew As QueryDef                 ' Постоянный запрос

    ' Открытие базы данных
    Set dbsSuppliersParts = OpenDatabase("ПоставщикиДетали.mdb")

    With dbsSuppliersParts
        ' Создание временного безымянного запроса методом
        ' Set querydef = object.CreateQueryDef (name, sqltext)
        ' здесь object - переменная - представляющая базу данных
        ' name - строка с именем нового запроса.
        ' Если name - пустая строка, запрос будет временным
        ' sqltext - строка с текстом SQL запроса

        Set qdfTemp = .CreateQueryDef("", _
            "SELECT * FROM Поставщики")
        ' Открытие набора записей и печать отчета.
        GetrstTemp qdfTemp      ' Вызов функции GetrstTemp
        ' Создание постоянного запроса.
        Set qdfNew = .CreateQueryDef("NewQueryDef", _
            "SELECT * FROM Детали")
        ' Открытие набора записей и печать отчета.
        GetrstTemp qdfNew
        ' Удаление нового запроса
        .QueryDefs.Delete qdfNew.Name
        .Close
    End With

End Sub

Операторы вида
Debug.Print

```

производят вывод в окно **Immediate**. Содержание этого окна с результатами работы программы показано на рис. 4.12. Видно, что временный запрос, которому в программе имя не было дано, получает имя по умолчанию **Temporary QueryDef**.

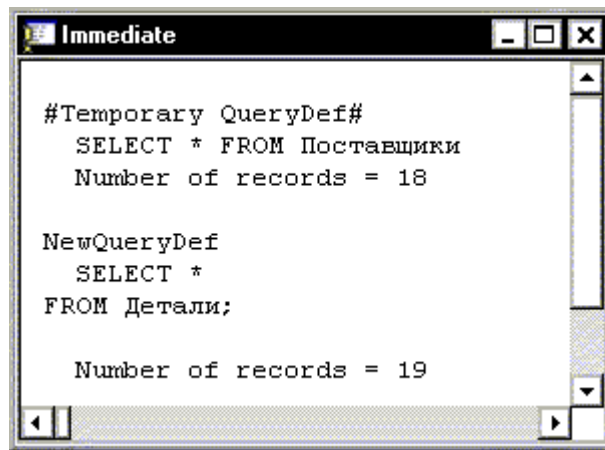


Рис. 4.12. Окно **Immediate** с результатами запросов в базе данных

4.5.2. Вывод содержимого запроса

В следующей процедуре CreateAndShowQueryDefX реализуется SQL-запрос к базе данных *ПоставщикиДетали*, включающий поставщиков и поставленные каждым поставщиком детали (см. п. 4.4.1):

```
SELECT Поставщик, Название
FROM Поставщики INNER JOIN
(Поставки INNER JOIN Детали On Поставки.КодДетали = Детали.КодДетали)
ON Поставщики.КодПоставщика = Поставки.КодПоставщика ORDER BY Поставщик;
```

В команду включено предложение ORDER BY Поставщик, обеспечивающее сортировку записей в запросе по полю *Поставщик*.

Вывод результатов запроса производится в окно **Immediate**.

```
Sub CreateAndShowQueryDefX()
    Dim dbsSuppliersParts As Database      ' Переменная - база данных
    Dim qdfSupPart As QueryDef             ' Запрос поставщики и поставленные детали
    Dim rstSupPart As Recordset             ' Переменная, представляющая набор записей

    ' Открытие базы данных
    Set dbsSuppliersParts = OpenDatabase("ПоставщикиДетали.mdb")

    ' Создание запроса и связывание с ним текста SQL-запроса
    Set qdfSupPart=dbsSuppliersParts.CreateQueryDef("ПоставщикиИ_ПоставленныеДетали", _
        "SELECT Поставщик, Название, Количество " & _
        " FROM Поставщики INNER JOIN" & _
        "(Поставки INNER JOIN Детали On Поставки.КодДетали = Детали.КодДетали)" & _
        " ON Поставщики.КодПоставщика = Поставки.КодПоставщика ORDER BY Поставщик")

    ' Создание набора записей по запросу
    Set rstSupPart = qdfSupPart.OpenRecordset(dbOpenSnapshot)
    With rstSupPart
        .MoveFirst                                'Перейти на первую запись
        While Not .EOF                             'Перебор записей в запросе
            ' Вывод полей запроса
            Debug.Print (.Fields("Поставщик") + " " + .Fields("Название"))
            .MoveNext                                'Переход к следующей дзаписи запроса
        Wend
    End With
    dbsSuppliersParts.QueryDefs.Delete (qdfSupPart.Name)      ' Удаление запроса
    dbsSuppliersParts.Close                                   ' Закрытие соединения с базой данных
```

End Sub

На рис. 4.13 показано окно **Immediate** с результатами работы процедуры CreateAndShowQueryDefX.

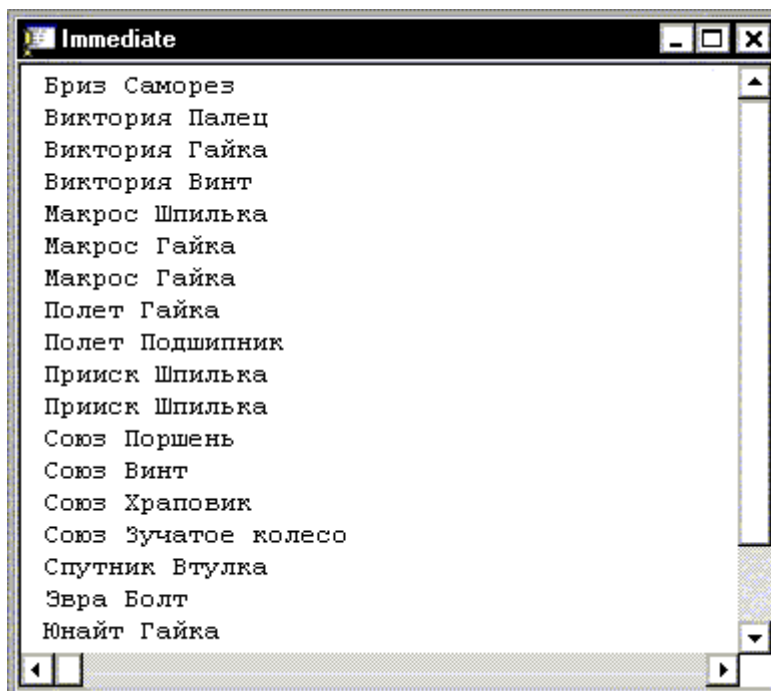


Рис. 4.13. Запрос с сортировкой по полю *Поставщик*

4.5.3. Сохранение запроса в таблице базы данных

Решим задачу, рассмотренную §2.9, с использованием запросов.

Напомним, что задача заключается в том, чтобы создать отчет, в котором должны быть перечислены поставщики и поставленные ими детали в виде одной строки с разделением названий деталей запятыми. Задача была решена путем создания таблицы с двум полями: *ПоставленныеДетали(Поставщик, Детали)*, поле которой *Детали* содержит нужную строку с названиями поставленных деталей, разделенными запятой. В приведенной в §2.9 программе вспомогательная таблица *ПоставленныеДетали* строится путем организации трех вложенных циклов, в которых перебираются строки основных таблиц базы данных: *Поставщики*, *Поставки* и *Детали* и заполняются строки таблицы *ПоставленныеДетали*.

Теперь будем заполнять таблицу *ПоставленныеДетали* используя запрос, в который включим поля *Поставщики.Поставщик* и *Детали.Название*, рассмотренный в процедуре CreateAndShowQueryDefX. Далее приводится текст процедуры FillTable_DeliveredPartsWithQuery, которая делает эту работу.

```
Public Sub FillTable_DeliveredPartsWithQuery()  
    Dim dbsSuppliersParts As Database  
  
    Set dbsSuppliersParts = CurrentDb          'Текущая база данных  
    ' Создание запроса на удаление записей из таблицы ПоставленныеДетали  
    Dim qdfDelDeliveredParts As QueryDef  
    Set qdfDelDeliveredParts = _  
        dbsSuppliersParts.CreateQueryDef("", _  
        "DELETE * FROM ПоставленныеДетали")
```

```

qdfDelDelivParts.Execute          ' Выполнение запроса на удаление

Dim qdfSupPart As QueryDef        ' Запрос поставщики и поставленные детали

'Наборы данных, представляющие:
Dim rstSupPart As Recordset      ' Набор записей из запроса
Dim rstDeliveredParts As DAO.Recordset ' таблицу ПоставленныеДетали
Set qdfSupPart =
dbsSuppliersParts.CreateQueryDef("ПоставщикиИ_ПоставленныеДетали", _
    "SELECT Поставщик, Название, Количество " & _
    " FROM Поставщики INNER JOIN" & _
    "(Поставки INNER JOIN Детали On Поставки.КодДетали = Детали.КодДетали)"
& _
    " ON Поставщики.КодПоставщика = Поставки.КодПоставщика " _
& " ORDER BY Поставщик")
'Присваивание наборам данных
Set rstDeliveredParts = dbsSuppliersParts.OpenRecordset("ПоставленныеДетали")
Set rstSupPart = qdfSupPart.OpenRecordset(dbOpenSnapshot)

Dim strSupplier As String        ' Строка для названия поставщика
Dim strParts As String          ' Строка для сохранения названий деталей
With rstSupPart
    While Not .EOF                ' Перебор записей в запросе
        strSupplier = .Fields("Поставщик") ' Запомнить название поставщика
        strParts = .Fields("Название") + ", " ' Название детали и запятая
        .MoveNext                ' Перейти к следующей записи запроса
        'Перебор записей для определенного поставщика
        Do While Not .EOF
            If strSupplier = .Fields("Поставщик") Then
                strParts = strParts + .Fields("Название") + ", " ' Назван. детали и запятая
                .MoveNext
            Else
                Exit Do            'Досрочный выход из цикла
            End If
        Loop
        With rstDeliveredParts    'Для таблицы ПоставленныеДетали
            .AddNew                'Добавляем новую запись
            .Fields("Поставщик") = strSupplier 'Заполняем поле Поставщик
            'Убираем последнюю запятую из строки
            strParts = Mid(strParts, 1, InStrRev(strParts, ",") - 1)
            .Fields("Детали") = strParts 'Заполняем поле Детали
            .Update                ' Сохранение добавленной записи в таблице
        End With
    Wend
End With
rstDeliveredParts.Close
dbsSuppliersParts.QueryDefs.Delete (qdfSupPart.Name) ' Удаление запроса
dbsSuppliersParts.Close ' Закрытие соединения с базой данных

```

End Sub

Литература

4.6. Основная

1. Дейт К. Введение в системы баз данных.– М.: Издательский дом «Вильямс», 2001.– 1072с.
2. Крёнке Д. Теория и практика построения баз данных.– СПб.: Питер, 2005.–859с.
3. Карпова Т.С. Базы данных: модели, разработка, реализация.– СПб.: Питер, 2002.– 304с.
4. Хомоненко А.Ф., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высш. уч. завед./ Под ред. проф. А.Д.Хомоненко.– СПб.: КОРОНА принт, 2002.– 672
5. Харрингтон Д.Л. Проектирование реляционных баз данных. Просто и доступно.–М.: Лори, 2000.– 230с.
6. Глушаков С.В., Ломотько Д.В. Базы данных.–Харьков: Фолио; М.: «Издательство АСТ», 2002.– 504 с.
7. Пасько В. Access 2000 (русифицированная версия).– К.: Издательская группа ВНУ, 1999.–384с.
8. Хелворсон М., Янг М. Эффективная работа с Microsoft Office 2000. – СПб.: Питер, 2000.– 1232 с.
9. Золотова С.И. Практикум по Access.– М.: Финансы и статистика, 2000.–144с.
10. Штайн Г. Access 2000. М.: Лаборатория базовых знаний, 2000.– 480с.
11. Андерсен В. Базы данных Microsoft Access. Проблемы и решения.– М.: Эком, 2001.– 384с.
12. Карпов Б. Microsoft Access 2000. Справочник.– СПб.: Питер, 2001.– 416с.
13. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных.- Н.Новгород: Изд-во ННГУ, 2004.-267с. (<http://www.unn.ru/rus/persons/shvetsov/>)

4.7. Дополнительная

14. Тимошок Т.В. Microsoft Access 2002. Краткое руководство. М.: Диалектика, 2004.– 272 с.
15. Microsoft Access 2002. Шаг за шагом.– М.: Эком 2002.– 352с.
16. Бекаревич. Ю., Пушкина Н. Самоучитель Microsoft Access 2003.– СПб.: БХВ-Петербург, 2004.– 738 с.
17. Хобракен Д. Microsoft Access 2000. Шаг за шагом.– М.: АСТ; Астрель, 2004.– 350 с.
18. Кузин А.В., Демин В.М. Разработка баз данных в системе Microsoft Access.– М.: Форум; Инфра-М 2005.– 224с.
19. Харитоновна И.А., Михеева В.Д. Microsoft Access 2000.– СПб.: БХВ-Петербург, 2001.– 1088с.