Федеральное агентство по образованию

Московский инженерно-физический институт
(государственный университет)

Т.В. Клецова, Н.В. Овсянникова, И.В. Прохоров

БАЗЫ ДАННЫХ

Лабораторный практикум

Рекомендовано УМО «Ядерные физика и технологии»
в качестве учебного пособия для студентов
высших учебных заведений

Москва 2008

УДК 004.65(076.5)
ББК 32.973-018.2я7
К48

Клецова Т.В., Овсянникова Н.В., Прохоров И.В. Базы данных: Лабораторный практикум. М.: МИФИ, 2008 – 132 с.

Пособие может служить основой как лабораторного практикума, так и самостоятельной работы. В нем на конкретных примерах изложены основные теоретические и практические сведения по проектированию, созданию, ведению баз данных, созданию приложений средствами СУБД Visual FoxPro.

Пособие предназначено для студентов, изучающих курс «Базы данных» в рамках специальностей «Прикладная информатика (в области международного сотрудничества, в социальных коммуникациях, в экономике)» и «Прикладная математика и информатика». Пособие также может быть полезно для студентов, обучающихся по другим специальностям.

Пособие подготовлено в рамках Инновационной образовательной программы.

Рецензент доцент кафедры компьютерных систем и технологий МИФИ Вавренюк А.Б.

ISBN 978-5-7262-0899-2

Содержание

Введение…….. 7
Общая постановка задачи на практические занятия и требования к программной реализации……..……..... 7
Установка каталога, используемого по умолчанию…… 9

Практическое занятие 1. Проектирование базы данных……. ...13
1.1. Концептуальное проектирование базы данных..…… ...13
1.1.1. Определение типов сущностей…… ...14
1.1.2. Определение типов связей…… ...14
1.1.3. Определение атрибутов и связывание их с типами сущностей и связей…… ...16
1.1.4. Определение доменов атрибутов…….. ...18
1.1.5. Определение атрибутов, являющихся потенциальными и первичными ключами……..... ...19
1.2. Логическое проектирование базы данных (для реляционной модели)…… ...20
1.2.1. Преобразование локальной концептуальной модели данных в локальную логическую модель…….. ...20
1.2.2. Определение набора отношений, исходя из структуры локальной логической модели данных…..... ...21
1.2.3. Проверка модели с помощью правил нормализации. ...22
1.2.4. Создание диаграмм «сущность-связь» логической модели данных…….. ...23
1.3. Физическое проектирование базы данных (с использованием реляционной СУБД)…….. ...24
1.3.1. Имена полей таблицы…… ...24
1.3.2. Типы данных…….. ...25
Заключение…… ...27
Вопросы для самоконтроля……..…… ...28

Практическое занятие 2. Работа с таблицами в СУБД Visual FoxPro …….. ...29
2.1. Работа с таблицами…… ...29
2.1.1. Создание таблиц…… ...29
2.1.2. Изменение структуры таблиц…….. ...35

2.1.3. Копирование таблиц…….. ...36
2.1.4. Удаление таблиц…… ...36
2.1.5. Использование рабочих областей…… ...36
2.2. Манипулирование данными в таблице …… ...38
2.2.1. Ввод данных……... ...38
2.2.2. Удаление записей……... ...38
2.2.3. Редактирование данных…… ...40
2.2.4. Фильтрация данных…… ...40
2.3. Работа с индексами…… ...42
2.3.1. Создание индексов…… ...42
2.3.2. Удаление индексов…… ...46
2.4. Реструктуризация базы данных…… ...47
2.4.1. Создание базы данных……... ...47
2.4.2. Работа с таблицами в базе данных……... ...49
2.4.3. Связи в базе данных……... ...51
2.4.4. Удаление базы данных…….. ...54
Заключение……...… ...54
Вопросы для самоконтроля……..…….. ...54

Практическое занятие 3. Программирование на языке FoxPro…… ...55
3.1. Создание программ……..…… ...55
3.2. Редактирование программ……...… ...56
3.3. Удаление программ……..……... ...56
3.4. Выполнение программ……..…….. ...56
3.5. Примеры программ……..…… ...56
Пример 1. Программа индексирования данных таблицы… ...56
Пример 2. Поиск данных в индексированной таблице…… ...58
Пример 3. Построение графиков…… ...59
Пример 4. Рисование линий…… ...60
Пример 5. Подсчет количества строк таблицы, удовлетворяющих некоторому условию……. ...61
Пример 6. Программа построения гистограммы…….. ...62
Заключение……...… ...66
Вопросы для самоконтроля……...…… ...66

Практическое занятие 4. Создание экранных форм, отчетов, этикеток и меню. Визуальное программирование…..... ...67

4.1. Работа с формами……... ...67
4.1.1. Создание формы…… ...67
4.1.2. Запуск формы…… ...75
4.1.3. Модификация формы…… ...77
4.1.4. Удаление формы…… ...80
4.2. Работа с отчетами……... ...81
4.2.1. Создание отчетов……... ...81
4.2.2. Печать отчета…….. ...87
4.2.3. Модификация отчета…… ...88
4.2.4. Удаление отчета…… ...91
4.3. Работа с этикетками……... ...91
4.3.1. Создание этикетки…… ...91
4.3.2. Печать этикетки…… ...94
4.3.3. Модификация этикетки…… ...94
4.3.4. Удаление этикетки…… ...94
4.4. Работа с меню…….……... ...95
4.4.1. Создание меню……... ...95
4.4.2. Запуск меню…… ...98
4.4.3. Клавиши быстрого доступа и заголовки пунктов меню…… ...98
4.4.4. Модификация меню……... .102
4.4.5. Удаление меню……... .102
Заключение…….. .103
Вопросы для самоконтроля…… .103

Практическое занятие 5. Структурированный язык запросов SQL…… .104
5.1. Общая характеристика языка…… .104
5.2. Основные операторы языка……... .105
5.2.1. Оператор создания таблицы…… .107
5.2.2. Оператор изменения структуры таблицы…… .108
5.2.3. Оператор создания представления……... .109
5.2.4. Оператор выборки записей……... .110
5.2.5. Оператор изменения записей…… .112
5.2.6. Оператор вставки новых записей…… .112
5.2.7. Оператор удаления записей…….. .113
5.2.8. Оператор создания временной таблицы…….. .113
Заключение…….. .114
Вопросы для самоконтроля…… .114

Список источников информации……... .115

Приложения……. .117
Приложение 1. Определения нормальных форм…….. .118
Приложение 2. Алгоритмы программ построения гистограмм.. .119
Приложение 3. Команды языка FoxPro, используемые в этом пособии…… .124
Приложение 4. Список расширений для именования различных типов файлов в Visual FoxPro……. .126
Приложение 5. Полный синтаксис команд SQL, поддерживаемых СУБД Visual FoxPro……..... .128

Введение

Целью практических занятий по курсу «Базы данных» является закрепление на практике основных понятий теории реляционных баз данных. Для этого предлагается разработать и реализовать в СУБД Visual FoxPro прототип автоматизированной информационной системы, отвечающей некоторым общим требованиям.

Выбор в качестве СУБД Visual FoxPro обусловлен тем, что эта СУБД имеет очень мощный и в то же время простой и интуитивно понятный графический интерфейс для выполнения любых действий по разработке приложения: от средств управления всем проектом до средств разработки конкретного окна формы или отчета. При создании любого объекта возможно использование мастера (Wizard), который позволяет автоматизировать процесс создания приложений.

В предлагаемом пособии на основе конкретных примеров изложены основные теоретические сведения по созданию и ведению баз данных и созданию приложений средствами СУБД Visual FoxPro 6.0. Заданной предметной областью в данном пособии для примера является разработка базы данных и программного комплекса для учета товаров в обувном магазине.

Команды встроенного языка программирования FoxPro и операторы SQL выделены в тексте шрифтом `Courier`, названия таблиц и полей таблиц – шрифтом Arial, заголовки окон и названия кнопок – шрифтом ***Arial (полужирный курсив)***.

Общая постановка задачи на практические занятия и требования к программной реализации

В результате выполнения задания должен быть спроектирован и программно реализован в среде Visual FoxPro прототип автоматизированной информационной системы (АИС), соответствующий предложенной предметной области. Интерфейс АИС должен представлять собой меню, реализующее функции ввода и коррекции данных, поиск данных по шаблону, отображение диаграмм, печать отчетов и выход. Общая схема программного комплекса представлена на рис. В1.

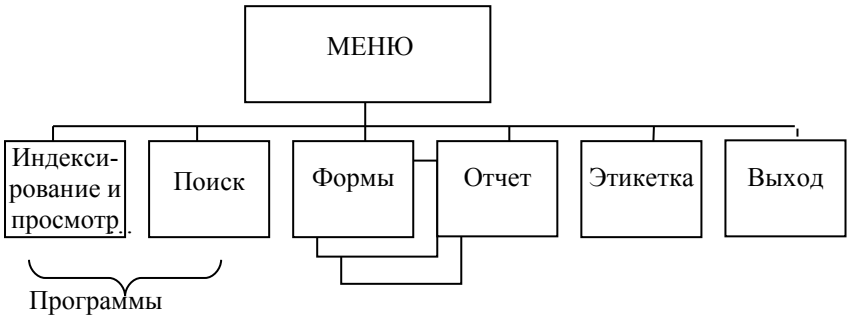


Рис. В1. Структурная схема программного комплекса

Порядок выполнения работы.

1. Получить от преподавателя вариант задания: описание предметной области, основные сущности, цель разработки базы данных.
2. Проанализировать сущности, допроектировать базу данных (построить отношения).
3. Построить в СУБД Visual FoxPro базу данных.
4. Заполнить таблицы данными.
5. Реализовать интерфейс, состоящий из форм, отчетов и этикеток в соответствии с требованиями к программной реализации.
6. Разработать программы поиска данных и построения гистограммы.
7. Разработать меню.
8. Протестировать и отладить разработанное программное обеспечение.
9. Оформить отчет.

Требования к программной реализации:

- АИС должна включать в себя, по крайней мере, две таблицы (не менее 5 полей и не менее 10 строк данных в каждой);
- написать не менее двух программ в соответствии с заданием;
- необходимо реализовать не менее трех форм (по одной для каждой таблицы и одну форму со связью «один-ко-многим»);
- если экранные формы создаются автоматически с помощью мастера форм (Wizard), то на одну из них необходимо добавить

хотя бы один дополнительный элемент (например, программную кнопку);
- необходимо реализовать, по крайней мере, один отчет (report) и не менее одной этикетки (label);
- основное меню должно содержать следующие возможности:
    - индексирование и просмотр данных таблиц;
    - ввод и коррекция данных;
    - поиск данных по шаблону;
    - печать отчетов;
    - печать этикеток;
    - выход в меню FoxPro.

По результатам выполнения практических занятий оформляется отчет. Отчет должен содержать:
- титульный лист;
- формулировку задания в предметной области;
- описание логической структуры проекта базы данных;
- описание физической структуры базы данных в СУБД Visual FoxPro;
- описание разработанного интерфейса (форм, отчетов, этикетки);
- тексты программ с комментариями;
- файлы данных (распечатать);
- заключение.

Установка каталога, используемого по умолчанию

Перед началом работы (чтобы знать месторасположение файлов и не потерять их) желательно создать свой каталог и задать этот каталог в качестве используемого по умолчанию.Visual FoxPro работает с файлами из одного каталога (если явно не указаны пути к файлам). После установки пакета рабочим становится каталог, в который установлен пакет, обычно `C:/program Files/Microsoft Visual Studio/Vfp98`.

Установить каталог, используемый по умолчанию, можно двумя способами: с помощью меню Visual FoxPro или с помощью команды.

Для задания каталога, используемого по умолчанию, с помощью меню Visual FoxPro следует выбрать пункт меню ***Tools*** → ***Options***, после чего откроется диалоговое окно ***Options***. На вкладке ***File Location*** выбрать строку ***Default Directory*** и нажать на кнопку ***Modify*** (рис. В2).

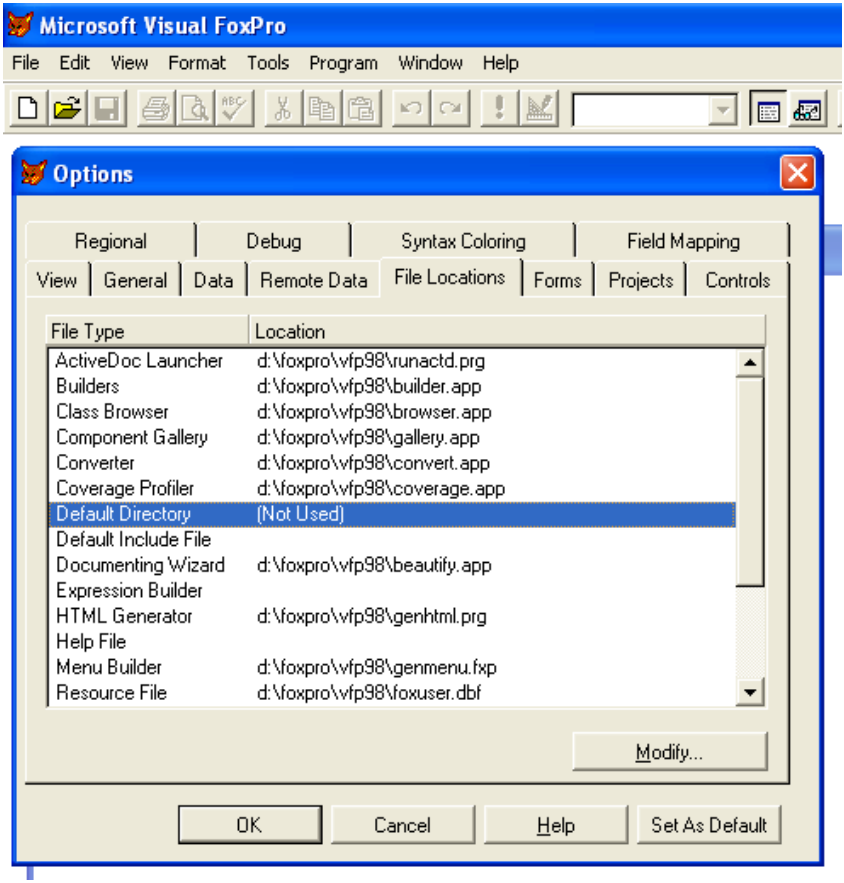


Рис. В2. Вкладка File Location диалогового окна Options

В диалоговом окне ***Change File Location*** (рис. В3) установить флажок ***Use default directory*** (поле ***Location of default directory*** для ввода каталога, используемого по умолчанию, станет доступным для изменения), ввести в поле ***Location of default directory*** путь до каталога по умолчанию или нажать на кнопку  (с тремя точками) и выбрать нужный каталог с помощью проводника.

После нажатия на кнопку ***OK*** выбранный каталог будет использоваться в текущем сеансе работы с Visual FoxPro.

После нажатия на кнопку ***Set As Default*** выбранный каталог будет использоваться Visual FoxPro по умолчанию.

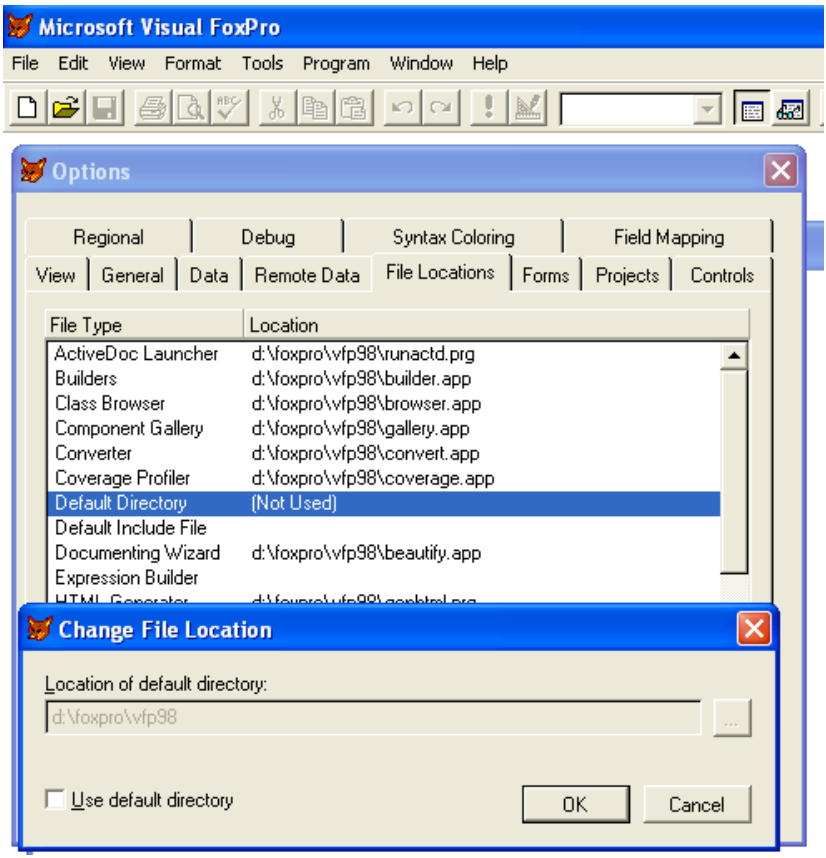


Рис. В3. Установка каталога, используемого по умолчанию

Для задания каталога, используемого по умолчанию, с помощью команды следует в командном окне набрать команду `SET DEFAULT TO [путь]`, где `[путь]` – полный путь к каталогу, который будет использоваться по умолчанию.

С помощью команды `SET DEFAULT TO ?` каталог, который будет использоваться по умолчанию, можно выбрать с помощью проводника.

После выполнения команды указанная директория будет использоваться по умолчанию в течение текущего сеанса работы с Visual FoxPro.

Практическое занятие 1
Проектирование базы данных

Цель работы: научиться создавать модели данных: определять сущности предметной области, их атрибуты, первичные и альтернативные ключи, устанавливать отношения между сущностями, разрабатывать концептуальную, логическую и физическую модели данных.

1.1. Концептуальное проектирование базы данных

База данных представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отражающих состояние объектов и их взаимосвязи в рассматриваемой предметной области [1]. Другими словами, база данных – это совокупность таблиц, связанных отношениями между собой.

Первый этап проектирования базы данных состоит в разработке концептуальных моделей данных для каждого из существующих типов пользователей создаваемого приложения (локальных концептуальных моделей). Представление пользователя включает в себя данные, необходимые конкретному пользователю для принятия решений или выполнения некоторого задания. Обычно представление пользователя отражает некоторую функциональную область в общем поле деятельности предприятия – например, производство, маркетинг, сбыт, управление кадрами или складской учет. Пользователь может быть как отдельным работником, так и группой лиц, которые будут непосредственно работать с создаваемым приложением [8].

Каждая локальная концептуальная модель данных включает:

- типы сущностей;
- типы связей;
- атрибуты;
- домены атрибутов;
- потенциальные ключи;
- первичные ключи.

1.1.1. Определение типов сущностей

Сначала необходимо определить основные объекты, которые могут интересовать пользователя. Эти объекты являются типами ***сущностей***, входящих в модель базы данных. Сущность – это абстракция множества объектов реального мира, в котором все предметы множества имеют одинаковые характеристики и согласованы с одним и тем же набором правил и линий поведения. Один из методов идентификации сущностей состоит в изучении спецификаций по выполнению конкретных функций пользователя на данном предприятии. Из этих спецификаций следует извлечь все используемые в них существительные или сочетания существительного и прилагательного. Для нашего случая – разработки базы данных учета товара в обувном магазине – можно выделить следующие сущности: «Товар», «Страна-производитель», «Количество товара», «Цена товара», «Цвет товара», «Адрес производителя», «Директор фирмы-производителя».

Затем среди выделенных сущностей выбираются самые крупные объекты или представляющие интерес концепции и исключаются все существительные, которые просто определяют другие объекты. Например, сущности «Адрес производителя» и «Директор фирмы-производителя» могут быть объединены в сводном объекте под названием «Производитель», тогда как сущности «Цена товара», «Цвет товара», «Количество товара» можно объединить в сущности под названием «Товар».

Таким образом, получается, что для нашего примера выделено только две сущности: «Товар» и «Производитель».

1.1.2. Определение типов связей

После выделения сущностей следующим этапом разработки концептуальной модели данных будет установление всех существующих между сущностями ***связей.*** Связь – это идентификатор требований, в соответствии с которыми сущность вовлекается в отношение. Отношения именуются с помощью глагола. Необходимо выбирать все выражения со словами-сущностями, в которых содержатся глаголы. Например: фирма-производитель *производит* товар. Нас интересуют только те связи между сущностями, которые необходимы для удовлетворения требований к проекту. В

большинстве случаев связи являются парными – другими словами, связи существуют только между двумя сущностями. Однако следует проявлять осторожность и тщательно проверять наличие в проекте комплексных связей, объединяющих более двух сущностей различных типов, а также рекурсивных связей, существующих между сущностями одного и того же типа [8].

Особое внимание следует уделять проверке того, были ли выделены *все* связи, явно или неявно присутствующее в спецификациях на проект. В принципе, каждую из возможных пар сущностей было бы полезно проверить на наличие между ними некоторой связи, что может оказаться чрезвычайно трудоемкой задачей. Но вообще отказываться от выполнения подобных проверок неразумно [8].

Установив связи, которые будут иметь место в создаваемой модели, необходимо определить кардинальность каждой из них. Каждая связь может иметь кардинальность либо «один-к-одному» (1:1), либо «один-ко-многим» (1:M), либо «многие-ко-многим» (M:N). Для нашего примера верно следующее:

- производитель может производить несколько товаров;
- один товар (с уникальным артикулом) может быть произведен только одним производителем.

Следовательно, кардинальность связи сущностей «Производитель» и «Товар» – «один-ко-многим» (1:M). Схема, представленная на рис. 1.1, называется ***диаграммой «сущность-связь»*** (или ER-диаграммой). ER-диаграмма является методом представления логической структуры базы данных в графическом виде для более простого и наглядного отображения основных компонентов конкретного проекта базы данных. Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных. Однако так как мы рассматриваем именно реляционные СУБД, то можно считать, что логическая модель данных для нас формулируется в терминах реляционной модели данных.

Существует четыре графических нотации диаграммы «сущность-связь»: Чена (Chen), IDEF1x, Мартина (Martin), Баркера (Barker). В данном лабораторном практикуме используется нотация Мартина, в которой сущности обозначены прямоугольниками,

имя связи указывается на линии, ее обозначающей, атрибуты записываются списком внутри прямоугольника сущности, первичный ключ подчеркивается.

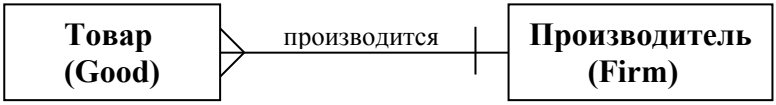


Рис. 1.1. Диаграмма “сущность-связь” на уровне сущностей

1.1.3. Определение атрибутов и связывание их с типами сущностей и связей

На следующем этапе предлагаемой методологии построения концептуальной модели необходимо выявить атрибуты сущностей, т.е. все данные, описывающие сущности и связи, выделенные в создаваемой модели базы данных. Воспользуемся тем же методом, который применялся для идентификации сущностей: выберем все существительные и содержащие их фразы, присутствующие в спецификациях на проект. Выбранное существительное представляет атрибут в том случае, если оно описывает свойство, качество, идентификатор или характеристику некоторой сущности или связи [8].

Самым простой метод выделения атрибутов – после идентификации очередной сущности или связи задать себе следующий вопрос: «Какую информацию требуется хранить о...». Каждому выявленному атрибуту следует присвоить осмысленное имя, понятное пользователям. В некоторых случаях может оказаться полезным попросить пользователей уточнить их требования к хранимой информации [8].

В нашем примере для сущности «Товар» можно выделить такие атрибуты, как: «Наименование товара», «Фирма-производитель», «Цена», «Цвет», «Количество». Для сущности «Производитель» – «Название фирмы», «Адрес», «Телефон», «ФИО директора», «№ банковского счета».

На рис. 1.2. приведена ER-диаграмма на уровне атрибутов.

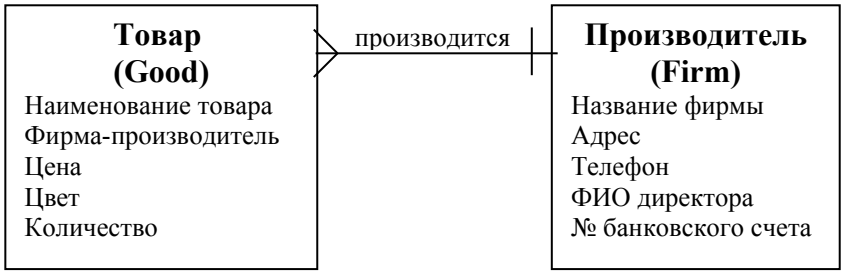


Рис. 1.2. Диаграмма “сущность-связь” на уровне атрибутов

Важно отметить, что каждый атрибут может быть либо ***простым***, либо ***составным***. Составные атрибуты представляют собой набор простых атрибутов. Например, атрибут «Адрес» может быть простым и представлять все элементы адреса как единое значение: «115409, Москва, Каширское ш., д.31». В другом варианте этот же атрибут может быть представлен как составной, т.е. состоящий из нескольких простых атрибутов, содержащих различные элементы адреса. В этом случае то же самое значение может быть разделено на такие атрибуты, как «Почтовый код» (115409), «Город» (Москва), «Улица» (Каширское ш.), «Дом» (31). Выбор способа представления адреса в виде простого или составного атрибута определяется требованиями, предъявляемыми к приложению пользователем. Если пользователь не нуждается в доступе к отдельным элементам адреса, то его целесообразно представить как простой атрибут. Но если пользователю потребуется независимый доступ к отдельным элементам адреса, то атрибут «Адрес» следует сделать составным, образованным из необходимого количества простых атрибутов.

Для нашего примера атрибут ФИО директора будет являться составным, так как при обращении в письме, например, необходимы только имя и отчество, а в почтовом адресе инициалы. Атрибут адрес предполагается использовать только как почтовый адрес, поэтому этот атрибут будет простым.

О каждом атрибуте в отчет помещаются следующие сведения:

- ***имя атрибута и его описание***;
- значение, принимаемое для атрибута ***по умолчанию*** (если таковое имеется);

- является ли атрибут ***обязательным*** (т.е. может ли он отсутствовать или иметь значение NULL);
- является ли атрибут ***составным*** и, если это так, из каких простых атрибутов он состоит;
- является ли данный атрибут ***вычисляемым*** и, если это так, какой метод следует использовать для вычисления его значения;
- является ли данный атрибут ***множественным***, т.е. может ли атрибут иметь одновременно несколько значений для одного и того же экземпляра сущности.

1.1.4. Определение доменов атрибутов

Доменом атрибута называется некоторый набор значений, элементы которого выбираются для присвоения значений атрибуту. Домены должны содержать следующие данные:

- набор допустимых значений для атрибута;
- сведения о размере и формате каждого из полей атрибутов.

Определим домены для каждого атрибута из нашего примера (табл. 1.1).

Таблица 1.1. Домены атрибутов

Название атрибута	Домен
Наименование товара	Любое строковое (текстовое) выражение, размером 15 символов
Фирма-производитель	Любое строковое (текстовое) выражение, размером 15 символов
Цена	Денежное выражение, больше нуля
Цвет	Любое строковое (текстовое) выражение, размером 8 символов
Количество	Числовое поле, целое, больше нуля
Адрес	Любое строковое (текстовое) выражение, размером 20 символов
Телефон	Числовое поле, длиной 7 символов, больше нуля
Фамилия директора	Любое строковое (текстовое) выражение, размером 15 символов
Имя директора	Любое строковое (текстовое) выражение, размером 10 символов

Продолжение табл. 1.1

Название атрибута	Домен
Отчество директора	Любое строковое (текстовое) выражение, размером 15 символов
№ банковского счета	Любое строковое (текстовое) выражение, размером 20 символов

1.1.5. Определение атрибутов, являющихся потенциальными и первичными ключами

На этом этапе для каждой сущности устанавливается потенциальный ключ (или ключи), после чего осуществляется выбор первичного ключа. ***Потенциальным ключом*** называется атрибут или минимальный набор атрибутов заданной сущности, позволяющий уникальным образом идентифицировать каждый ее экземпляр. Для некоторых сущностей возможно наличие нескольких потенциальных ключей. В этом случае среди них нужно выбрать один ключ, который будет называться ***первичным ключом***. Первичный ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Все остальные потенциальные ключи будут называться ***альтернативными ключами***. При выборе первичного ключа среди нескольких потенциальных ключей будем руководствоваться приведенными ниже рекомендациями [8]:

- выбирается потенциальный ключ с минимальным набором атрибутов;
- выбирается тот потенциальный ключ, вероятность изменения значений которого минимальна;
- выбирается тот потенциальный ключ, который имеет минимальную вероятность потери уникальности значений в будущем;
- выбирается потенциальный ключ, значения которого имеют минимальную длину (в случае текстовых атрибутов);
- выбирается потенциальный ключ, с которым будет проще всего работать (с точки зрения пользователя).

В нашем случае потенциальными ключами будут являться:

- для сущности «Товар»: «Название товара» + «Фирма-производитель» + «Цвет»;
- для сущности «Производитель»: «Название фирмы».

Среди потенциальных ключей выберем первичные ключи:

- для сущности «Товар»: так как потенциальный ключ сущности «Товар» сложен (состоит из 3 атрибутов), то целесообразно создать «искусственное» ключевое поле, например «Уникальный индекс товара»;
- для сущности «Производитель»: «Название фирмы».

1.2. Логическое проектирование базы данных (для реляционной модели)

1.2.1. Преобразование локальной концептуальной модели данных в локальную логическую модель

На этом этапе структура данных преобразуется в такую форму, которая не вызовет затруднений при реализации в СУБД.

На данном этапе выполняются следующие действия.

- ***Удаление связей типа M:N***. Если в концептуальной модели присутствуют связи типа M:N, то их следует устранить путем определения некоторой промежуточной сущности. Связь типа M:N заменяется двумя связями типа 1:M, устанавливаемыми со вновь созданной сущностью.
- ***Удаление сложных связей***. Сложной называется связь, существующая между тремя и больше типами сущностей. Если в концептуальной модели присутствует сложная связь, ее следует устранить с помощью промежуточной сущности. Сложная связь заменяется необходимым количеством бинарных связей типа 1:M, устанавливаемых со вновь созданной сущностью.
- ***Удаление рекурсивных связей***. Рекурсивными называются такие связи, в которых сущность некоторого типа взаимодействует сама с собой. Если концептуальная модель содержит рекурсивные связи, они должны быть устранены посредством определения некоторой промежуточной сущности.
- ***Удаление связей с атрибутами***. Если в концептуальной модели присутствуют связи, имеющие собственные атрибуты, они должны быть преобразованы путем создания новой сущности.

•***Удаление множественных атрибутов***. Если в концептуальной модели присутствует множественный атрибут, его следует преобразовать путем определения новой сущности.

•***Перепроверка связей типа 1:1.*** В процессе определения сущностей могли быть созданы две различные сущности, которые на самом деле представляют один и тот же объект в предметной области приложения. В подобном случае следует объединить эти две сущности в одну. Если первичные ключи объединяемых сущностей различны, необходимо выбрать один из них в качестве первичного, а другой указать как альтернативный ключ.

•***Удаление избыточных связей***. Связь является избыточной, если одна и та же информация может быть получена не только через нее, но и с помощью другой связи. Всегда следует стремиться создавать минимальные модели данных, и поэтому, если избыточная связь не является очевидно необходимой, ее следует удалять. Установить, что между двумя сущностями имеется больше одной связи, довольно просто. Однако из этого еще не следует, что одна из двух связей обязательно является избыточной, поскольку обе они могут представлять различные объединения, реально существующие в организации.

В нашем примере связь одна, тип связи – «один-ко-многим», сложных, избыточных, рекурсивных и связей с атрибутами нет, множественных атрибутов нет (предполагается, что телефон и банковский счет у фирмы может быть только один), поэтому этот этап пропускается.

1.2.2. Определение набора отношений, исходя из структуры локальной логической модели данных

На данном этапе необходимо на основе созданных локальных логических моделей данных определить наборы отношений, необходимые для представления сущностей и связей, входящих в представления отдельных пользователей о предметной области приложения.

Связи, которые сущность имеет с другими типами сущностей, представляются с помощью механизма ***первичных и внешних ключей***. Для принятия решения о том, откуда взять и куда поместить значения атрибута (атрибутов) внешнего ключа, предварительно следует установить, какая из участвующих в связи сущно-

стей является родительской, а какая – дочерней. ***Родительской*** считается сущность, которая передает копию набора значений своего первичного ключа в отношение, представляющее ***дочернюю сущность***, где эти значения будут играть роль внешнего ключа.

Для каждой присутствующей в логической модели данных бинарной связи типа 1:1, установленной между сущностями E1 и E2, необходимо переслать атрибуты первичного ключа сущности E1 в отношение, представляющее сущность E2.

Для каждой бинарной связи типа 1:M, установленной в логической модели данных между сущностями E1 и E2, необходимо переслать копию атрибутов первичного ключа сущности E1 в отношение, представляющее сущность E2, где они будут играть роль внешнего ключа. Сущность, представляющая «единичную» сторону связи определяется как родительская, а сущность, представляющая «множественную» сторону, – как дочерняя. Для представления данной связи необходимо скопировать первичный ключ родительской сущности в отношение, представляющее дочернюю сущность, где этот ключ должен быть описан как внешний.

В нашем случае родительской является сущность «Производитель», а дочерней – «Товар», связь между сущностями – «один-комногим», следовательно, в сущности «Товар» должен быть атрибут, описывающий фирму-производителя. В нашем примере такой атрибут уже предусмотрен, это атрибут «Фирма-производитель», он будет являться внешним ключом.

1.2.3. Проверка модели с помощью правил нормализации

Нормализация используется для улучшения модели данных для того, чтобы модель удовлетворяла различным ограничениям, позволяющим исключить нежелательное дублирование данных. Нормализация гарантирует, что полученная в результате ее применения модель данных будет наилучшим образом отображать особенности использования информации на предприятии, не содержать противоречий, иметь минимальную избыточность и максимальную устойчивость.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);

- третья нормальная форма (3NF);
- нормальная форма Бойса–Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Таким образом, каждая нормальная форма является в некотором смысле более ограниченной, но и более *желательной*, чем предшествующая. Это связано с тем, что (N+1)-я нормальная форма не обладает некоторыми непривлекательными особенностями, свойственным N-й нормальной форме. Общий смысл дополнительного условия, налагаемого на (N+1)-ю нормальную форму по отношению к N-й нормальной форме, состоит в исключении этих непривлекательных особенностей. Определения нормальных форм приведены в приложении 1.

В нашем примере база данных приведена к первой нормальной форме (так как ни одна из строк таблиц не содержит в любом своем поле более одного значения, т.е. все значения атрибутов атомарны), ко второй нормальной форме (так как нет атрибутов, частично зависящих от ключевого поля) и к третьей нормальной форме (так как нет транзитивных зависимостей).

1.2.4. Создание диаграмм «сущность-связь» логической модели данных

После всех преобразований итоговая логическая модель базы данных представлена на рис. 1.3.

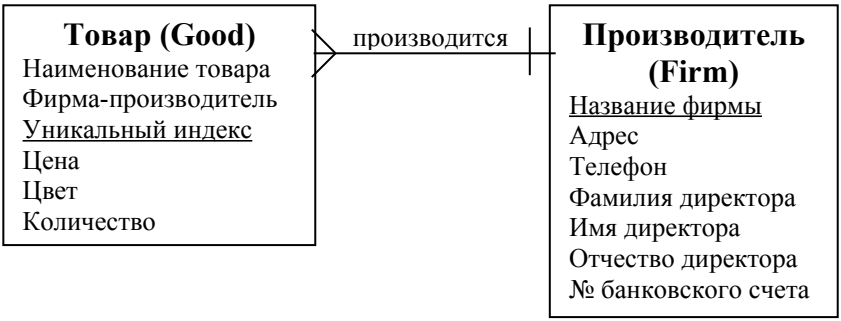


Рис. 1.3. Итоговая логическая модель базы данных учета товара в обувном магазине

1.3. Физическое проектирование базы данных (с использованием реляционной СУБД)

Физическое проектирование заключается в переносе логической модели данных в выбранную СУБД (в нашем случае в СУБД Visual FoxPro).

1.3.1. Имена полей таблицы

Имя поля таблицы в СУБД Visual FoxPro может состоять из 254 символов для таблиц, включенных в базу данных, и 10 символов для свободных таблиц (не включенных в базу данных). В имени поля можно применять буквы, цифры и знак подчеркивания. Использование знаков препинания, специальных символов и пробелов в имени поля не рекомендуется. Также имя поля не должно начинаться с цифры или знака подчеркивания [2]. Выберем имена для сущностей и атрибутов логической модели данных (табл. 1.2 и табл. 1.3).

Таблица 1.2. Таблица Goods

Наименование поля	Комментарии
Good_name	Наименование товара
Firm	Фирма-производитель
Index	Уникальный индекс на складе
Price	Цена
Colour	Цвет
Amount	Количество

Таблица 1.3. Таблица Firms

Наименование поля	Комментарии
Firm_name	Название фирмы
Address	Адрес
Telephone	Телефон
Director_f	ФИО директора
Director_n	
Director_o	
Account	№ банковского счета

1.3.2. Типы данных

После определения имен сущностей и их атрибутов необходимо решить вопрос о том, каким типом данных будет представляться тот или иной атрибут (поле таблицы). Выбор типа данных зависит от домена атрибута и влияет на суммарный объем БД, скорость поиска, допустимые операции с этим атрибутом и т.д. Таким образом, выбирая тип для того или иного атрибута, необходимо учитывать следующее: какие значения может принимать тот или иной атрибут, какие операции будут выполняться с данным атрибутом, сколько физического места займет одно значение для атрибута. Возможные типы данных для полей таблиц, поддерживаемые СУБД Visual FoxPro, приведены в табл. 1.4 [2, 3].

Таблица 1.4. Типы данных

Тип	**Диапазон**	**Объем памяти, байт**	**Описание**
Array			Массив данных некоторого типа
Double	от+/-4.94065648541247E-324 до +/-1.79769313486232E+308	8	Число с плавающей точкой двойной точности
Character	Любые символы	1–254	Текстовая (символьная) строка
Date	от 01/01/100 до 12/31/9909	8	Дата
Float	от-0,9999999999x10^{+19} до 0,9999999999x10^{+20}	8	Число с плавающей точкой
General	Определяется доступной памятью	4 (в dbf)	Ссылка на OLE-объект (для хранения объектов различных сред (графических объектов, текстовых файлов и др.))
Integer	- 2147483647 до 2147483646	4	Число целое
Logical	Истина ({T,t,Y,y}), Ложь ({F,f,N,n})	1	Логическое значение
Memo	Определяется доступной памятью (в dbf)	4	Ссылка на примечание
Numeric	от-0,9999999999x10^{+19} до 0,9999999999x10^{+20}	8	Число с фиксированной точкой целое или дробное
DateTime	от 01/01/1000 до 12/31/9999 и от 00:00:00 утра до 23:59:59 вечера	8	Дата и время
Currency	от -22337203685477,5807 до 922337203685477,5807	8	Денежное значение

Для рассматриваемого примера данные будут иметь следующий тип и размер (табл. 1.5 и табл. 1.6):

Таблица 1.5. Типы данных для таблицы Goods

Наименование поля	Тип	Примечание
Good_name	Character, 15	Текстовое поле длиной 15 символов
Firm	Character, 15	Текстовое поле длиной 15 символов
Index	Numeric, 2, 0	Числовое поле длиной 2 цифры
Price	Currency	Денежный тип
Colour	Character, 8	Текстовое поле длиной 8 символов
Amount	Integer	Целое число

Таблица 1.6. Типы данных для таблицы Firms

Наименование поля	Тип	Примечание
Firm_name	Character, 15	Текстовое поле длиной 15 символов
Address	Character, 20	Текстовое поле длиной 20 символов
Telephone	Numeric, 7, 0	Числовое поле длиной 7 цифр
Director_f	Character, 15	Текстовое поле длиной 15 символов
Director_n	Character, 10	Текстовое поле длиной 10 символов
Director_o	Character, 15	Текстовое поле длиной 15 символов
Account	Character, 20	Текстовое поле длиной 20 символов

Заключение

В ходе лабораторной работы студенты должны научиться разрабатывать модели данных (концептуальную, логическую, физическую): определять сущности, выявлять связи между ними, определять атрибуты сущностей, первичные и альтернативные ключи, строить схему реляционной базы данных.

Вопросы для самоконтроля

1. В чем отличие концептуальной, логической и физической моделей базы данных?
2. Что такое «сущность»?
3. Для чего определяются связи между сущностями? Какие типы связей Вы знаете?
4. Что такое атрибут?
5. Что включает в себя понятие «домен атрибута»?
6. Что такое ER-диаграмма?
7. Что такое первичный ключ? Потенциальный ключ?
8. Для чего проводится нормализация отношений?
9. Дайте определения 1-й, 2-й и 3-й нормальным формам.
10. Как представляются отношения в реляционной базе данных?
11. Какие типы данных поддерживаются СУБД Visual FoxPro?

Практическое занятие 2
Работа с таблицами в СУБД Visual FoxPro

Цель работы: научиться работать с таблицами в среде Visual FoxPro: создавать таблицы, объединять их в базу данных, добавлять или удалять записи из таблицы, создавать индексы и осуществлять отбор данных.

2.1. Работа с таблицами

2.1.1. Создание таблиц

Visual FoxPro, как и любая другая реляционная СУБД, хранит данные в таблицах. Таблицы можно создать двумя способами: через меню среды и через командное окно (рис. 2.1).

2.1.1.1. Создание таблиц через меню

Если таблица создается через меню, то следует выбрать пункт меню ***File→New***, в открывшемся диалоговом окне ***New*** (рис. 2.2) выбрать переключатель ***Table*** для создания таблицы и нажать кнопку ***New File***.

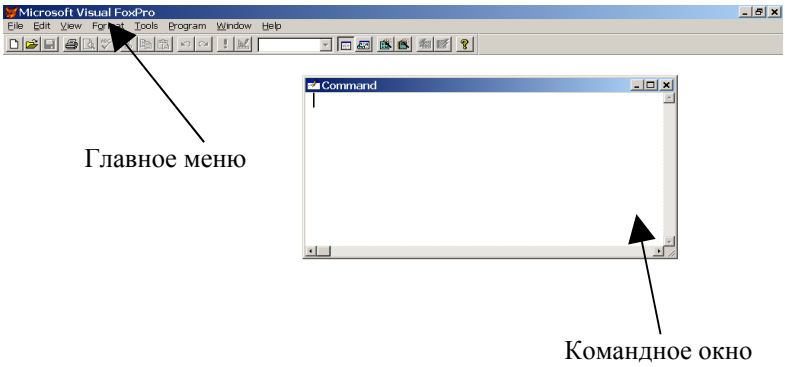


Рис. 2.1. Главное окно Visual FoxPro

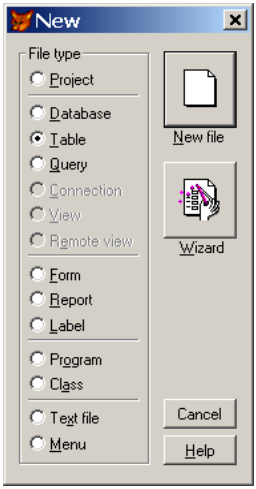


Рис. 2.2. Создание нового объекта в Visual FoxPro

При выборе мастера (нажатии кнопки ***Wizard*** в диалоговом окне ***New***, см. рис. 2.2) предлагается диалог, с помощью которого можно создать таблицы по имеющимся в FoxPro образцам таблиц (Accounts, Customers и т.д.).

Рассмотрим создание таблицы без применения мастера создания таблиц. После нажатия кнопки ***New File*** (см. рис. 2.2) требуется задать имя создаваемой таблицы, после чего открывается окно конструктора таблиц ***Table Designer*** (рис. 2.3).

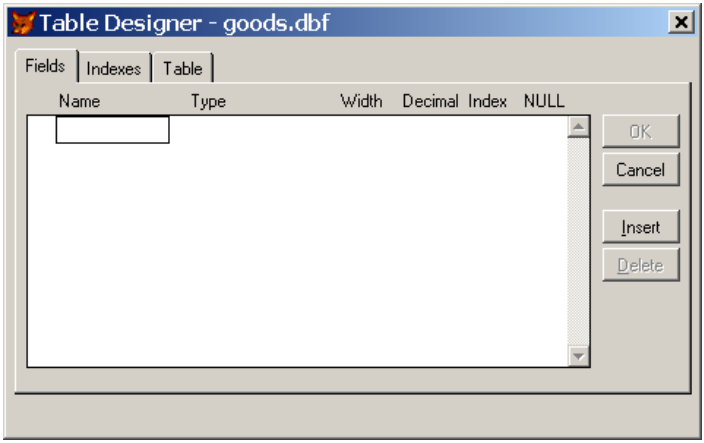


Рис. 2.3. Окно конструктора таблиц. Вкладка ***Fields***

Окно конструктора таблиц содержит три вкладки: ***Fields*** – для определения полей таблицы, ***Indexes*** – для определения индексов и ***Table*** – информационная вкладка.

Назначение кнопок в конструкторе таблиц следующее [2]:

OK – сохраняет структуру таблицы;

Cancel – отменяет проведенные изменения;

Insert – вставляет новое поле (новый атрибут) таблицы (для вкладки ***Fields***) или новый индекс (для вкладки ***Indexes***);

Delete – удаляет указанное поле (атрибут) таблицы (для вкладки ***Fields***) или указанный индекс (для вкладки ***Indexes***).

На вкладке **Fields** конструктора таблиц необходимо ввести названия полей (атрибутов) таблицы в столбце ***Name*** в соответствии с правилами, указанными в п.1.3.1, задать тип данных (см. табл. 1.4) для каждого поля в столбце **Type** (по умолчанию стоит тип данных Character), длину поля в столбцы ***Width*** и ***Decimal*** (***Width*** – общее число символов, ***Decimal*** – число символов после запятой, опция ***Decimal*** доступна только для полей типа Numeric). Столбец ***Index*** задает индекс (ascending – индекс по возрастанию, descending – индекс по убыванию), он автоматически сохраняется с типом Regular и с именем поля, по которому создан индекс (см. п.2.3) и отражается на вкладке ***Indexes***. Если индекс не определен, то Visual FoxPro при помощи данной опции автоматически создает его. Столбец ***Null*** предназначен для указания, допускается ли пустое значение в данном поле. Маленькие кнопки, расположенные слева от имен полей (), служат для изменения порядка отображения полей при просмотре таблицы.

На вкладке ***Indexes*** конструктора таблиц (рис. 2.4) отображаются все индексы, созданные по данной таблице.

В столбце ***Order*** вкладки ***Indexes*** конструктора таблиц отображается порядок индекса (ascending – возрастающий, descending – убывающий), в столбце ***Name*** – имя индекса, в столбце ***Type*** – тип индекса (см. табл. 2.1., с.42), в столбце ***Expression*** – поле или поля, по которым строится индекс.

На вкладке ***Table*** конструктора таблиц (рис. 2.5) отображается описание таблицы, т.е. адрес, по которому она сохранена (Table file), количество записей (Records), количество полей (Fields), размер записи в таблице (Length).

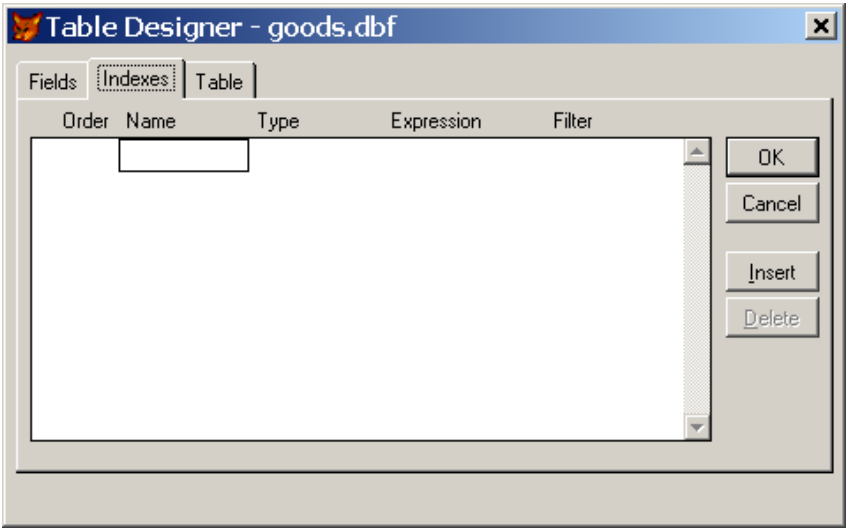


Рис. 2.4. Окно конструктора таблиц. Вкладка ***Indexes***

Рис. 2.5. Окно конструктора таблиц. Вкладка ***Table***

Для рассматриваемого примера на вкладке ***Fields*** определим названия столбцов таблицы goods и типы данных полей (рис. 2.6).

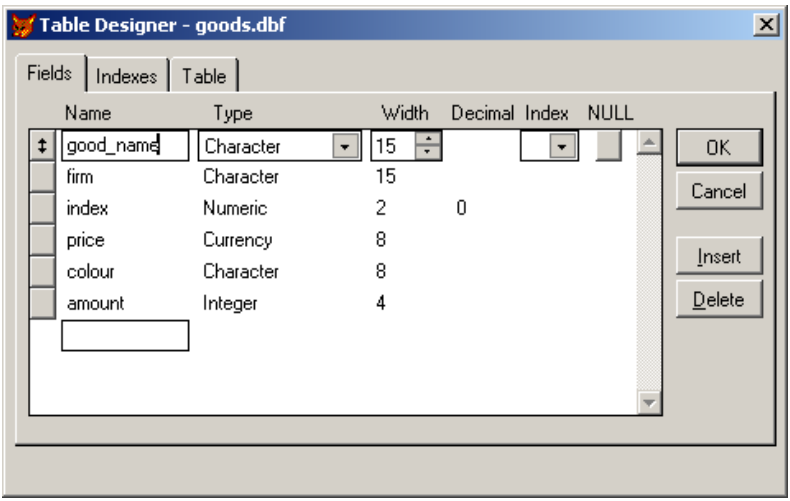


Рис. 2.6. Создание таблицы goods

Заполнив вкладку ***Fields***, нажимаем кнопку ***OK***, после чего предлагается немедленно начать заполнение таблицы (рис. 2.7).

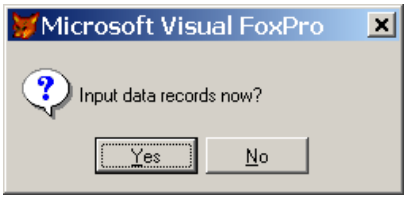


Рис. 2.7. Запрос на ввод данных

При нажатии кнопки **Yes** открывается окно для ввода значений атрибутов (рис. 2.8). По окончании ввода данных закрываем окно ввода (нажав).

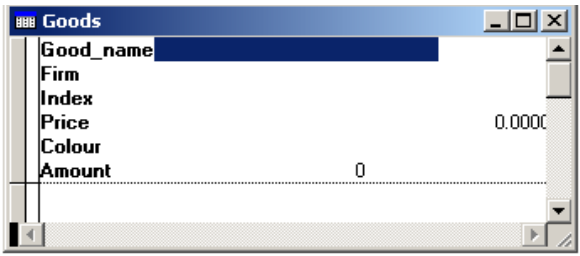


Рис. 2.8. Окно для ввода значений атрибутов

При отказе от немедленного заполнения таблицы (при нажатии кнопки ***No***, см. рис. 2.7.) можно вернуться к этапу ввода данных позже (см. п.2.2.1).

2.1.1.2. Создание таблиц через командное окно

Командное окно (см. рис. 2.1) предназначено для ввода команд на языке программирования Visual FoxPro и открывается автоматически при запуске Visual FoxPro или с помощью пункта меню ***Window→Command Window*** (или одновременным нажатием клавиш ***Cntr+F2***). Ввод любой команды завершается нажатием клавиши ***Enter***.

Таблицу через командное окно можно создать, выполнив команду `CREATE`. Для этого необходимо в командном окне написать команду `CREATE` и нажать клавишу ***Enter***. На экране появится диалоговое окно ***Create*** (рис. 2.9).

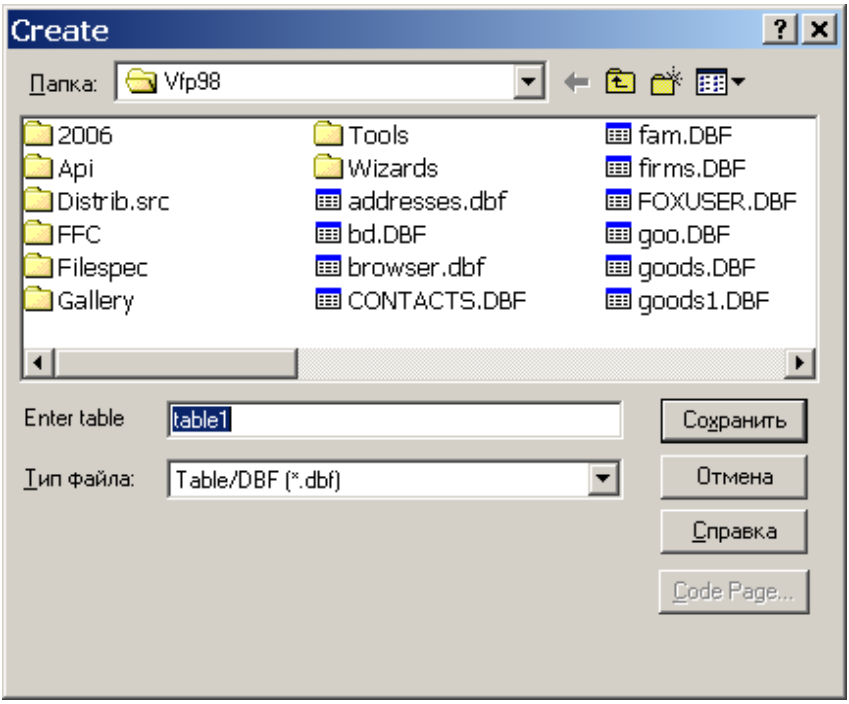


Рис. 2.9. Окно Create

В окне ***Create*** следует выбрать папку. Если предварительно не была установлена папка по умолчанию (например, с помощью команды `SET DEFAULT TO <имя_папки>`), открывается папка, в которой установлен пакет Visual FoxPro. В поле ***Enter table*** следует указать имя файла таблицы и нажать на кнопку ***Сохранить***. При создании таблицы следует присваивать ей имя, отражающее существо хранимой информации, но в рамках правил операционной системы Windows [2, 4]. После сохранения имени таблицы (см. рис. 2.9) откроется окно ***Table Designer*** – конструктор таблиц (см. рис. 2.3).

Если выполнить команду `CREATE <имя_таблицы>`, то таблица с указанным именем сохранится в папку по умолчанию и сразу откроется окно конструктора таблиц ***Table Designer*** (см. рис. 2.3).

Например, для создания таблицы фирм в командном окне введем команду `CREATE FIRMS`, после чего открывается конструктор таблицы (см. рис. 2.3). Дальнейшие действия аналогичны описанным выше для таблицы goods (см. п.2.1.1.1).

2.1.2. Изменение структуры таблиц

Если при вводе данных выявляются ошибки проектирования структуры таблицы (например, количества символов, выделенных для названия фирмы, не хватает), исправить структуру таблицы можно с помощью команды `MODIFY STRUCTURE`, написанной в командном окне:

```
SELECT <имя_таблицы>  && если таблица открыта[1]
USE <имя_таблицы>  &&если таблица не открыта
MODIFY STRUCTURE.
```

По этой команде откроется окно конструктора таблиц ***Table Designer*** (см. рис. 2.3–2.5), в котором можно сделать соответствующие исправления.

[1] *&&* – здесь и далее символ начала комментария в командной строке или строке программы (т.е. текст, следующий за этими символами, не интерпретируется Visual FoxPro как команда).

2.1.3. Копирование таблиц

Для копирования таблицы необходимо сначала ее открыть. Это можно сделать выбрав пункт меню ***File→Open*** или выполнив команду `USE <имя_таблицы>` из командном окне. Копирование осуществляется командой

```
COPY TO <имя_новой_таблицы>
```

или выбором пункта меню ***File→Save As…***.

2.1.4. Удаление таблиц

Для удаления таблицы необходимо выполнить команду `DROP TABLE <имя_таблицы>.`

Еще один способ удаления таблицы – выполнить команду `DELETE FILE <имя_таблицы>.dbf.`

В результате выполнения любой из этих команд файл таблицы будет удален с диска.

2.1.5. Использование рабочих областей

Рабочие области используются для одновременной работы с несколькими таблицами. Рабочая область – это виртуальный «рабочий стол», в одной рабочей области может быть открыта только одна таблица. Visul FoxPro имеет 32767 рабочих областей. Открывая таблицы в различных рабочих областях и переключаясь затем между этими рабочими областями (с помощью команды `SELECT`), можно одновременно работать сразу с несколькими таблицами [2, 3].

По умолчанию таблица открывается в текущей рабочей области, при этом уже открытая в этой рабочей области таблица закрывается. Например, после последовательного выполнения команд:

```
USE TABLE1
USE TABLE2
USE TABLE3
```

в результате будет открыта только таблица table3, так как при открытии таблицы table2 будет одновременно закрыта таблица table1, а при открытии таблицы table3 будет одновременно закрыта таблица table2.

Для того, чтобы открыть таблицу в рабочей области, отличной от текущей, нужно либо выбрать определенную рабочую область

(SELECT <номер рабочей области>), либо выбрать свободную рабочую область с наименьшим номером (SELECT 0). Например, в результате выполнения команд:

```
USE TABLE1 IN 1
USE TABLE2 IN 2
USE TABLE3 IN 3
```

будут открыты сразу три таблицы, каждая в своей рабочей области.

Конструкция USE TABLE1 IN 1 равносильна выполнению двух команд:

```
SELECT 1
USE TABLE1,
```

поэтому для открытия трех таблиц в разных рабочих областях можно написать следующие команды:

```
SELECT 1              SELECT 0
USE TABLE1            USE TABLE1
SELECT 2              SELECT 0
USE TABLE2     или     USE TABLE2
SELECT 3              SELECT 0
USE TABLE3            USE TABLE3
```

Открытой таблице Visual Fox Pro назначает псевдоним – имя, служащее для ссылок на эту таблицу. По умолчанию псевдоним имеет такое же имя, что и открытая таблица. Для переключения между рабочими областями следует ввести команды:

- SELECT <номер_рабочей_области>, если точно известно в какой рабочей области открыта таблица,
- SELECT <имя_таблицы>, если псевдоним соответствует имени таблицы (по умолчанию) или
- SELECT <псевдоним>, если псевдоним задан и не соответствует названию таблицы, открытой в данной рабочей области.

Команда USE без имени таблицы закрывает таблицу в текущей рабочей области (если рабочая область не указана) или в указанной с помощью команды SELECT рабочей области.

Для нашего примера сделаем следующее. Сделаем активной рабочую область № 1 командой SELECT, и в ней откроем таблицу goods (команды вводим в командном окне, см. рис. 2.1):

```
SELECT 1
USE GOODS.
```

Теперь просмотрим содержимое поля firm_name таблицы goods:

```
BROWSE FIELDS FIRM_NAME.
```

В рабочей области №2 откроем таблицу firms и начнем вводить в нее данные:

```
SELECT 2
USE FIRMS
APPEND.
```

Таким образом, с помощью команды SELECT можно переключаться между таблицами и работать с ними.

2.2. Манипулирование данными в таблице

2.2.1. Ввод данных

Для ввода новой строки (пустой) можно написать в командном окне команду APPEND BLANK или в окне просмотра содержимого таблицы (с помощью команды BROWSE) нажать сочетание клавиш ***Cntr+Y***. Команда APPEND без аргументов открывает окно (рис. 2.10), в котором можно вводить данные в таблицу, причем после заполнения строки добавляется новая строка.

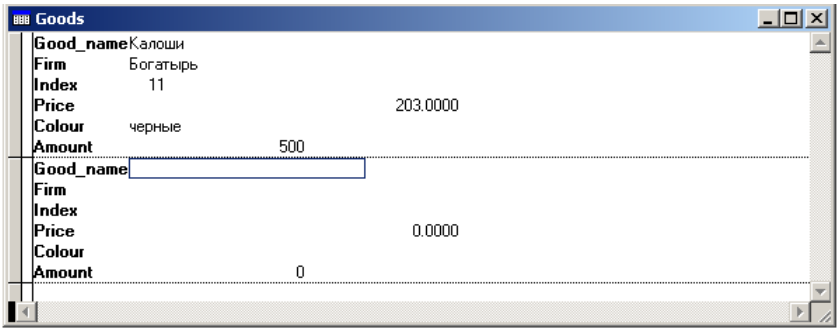


Рис. 2.10. Окно для добавления записей в таблицу goods

2.2.2. Удаление записей

Для удаления записи (строки) из таблицы необходимо сначала пометить эту запись на удаление, а затем физически удалить ко-

мандой PACK или через пункт меню ***Table→Remove Deleted Records***.

Чтобы пометить запись на удаление, нужно выполнить одно из следующих действий:

- для текущей записи (строки) нажать сочетание клавиш ***Cntr+T*** или выполнить команду DELETE;
- выполнить команду DELETE FOR <условие>, которая помечает на удаление все записи, удовлетворяющие указанному условию,
- щелкнуть левой кнопкой мыши на маркере удаления (крайний левый столбец в окне browse).

Записи, помеченные на удаление, в режиме просмотра содержимого таблицы в зависимости от установок либо не показываются, либо показываются, но помеченные черным перед первым полем записи (на маркере удаления) (рис. 2.11).

Если команда PACK не выполнена, то при нажатии сочетания клавиш ***Cntr+T*** в режиме просмотра содержимого таблицы для помеченной на удаление записи пометка на удаление будет отменена. Еще один способ отменить пометку на удаление текущей строки – выполнить команду RECALL.

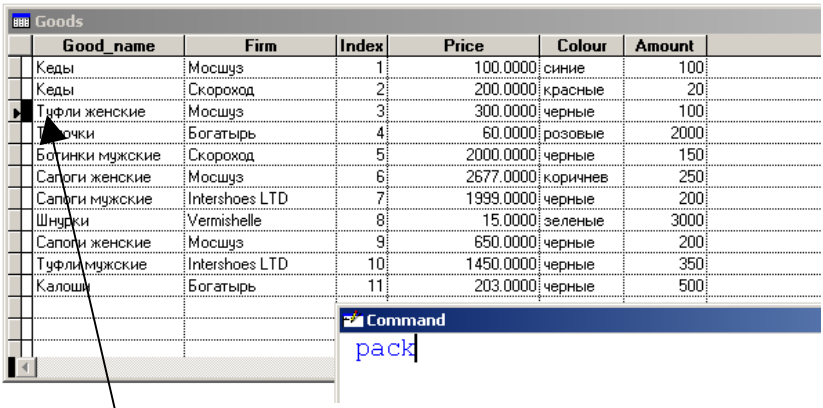


Рис. 2.11. Удаление строки (безвозвратно)

2.2.3. Редактирование данных

Редактирование отдельных значений полей записи осуществляется при просмотре содержимого таблицы (команда BROWSE).

Если необходимо заменить значение одного атрибута, то можно для текущей строки выполнить команду

```
REPLACE <имя_атрибута>;
   WITH <новое_значение_атрибута>
```

или для всех строк, удовлетворяющих заданному условию выполнить команду

```
REPLACE <имя_атрибута>;
   WITH <новое_значение_атрибута> FOR <условие>.
```

В результате выполнения следующих команд в таблицу будет добавлена новая пустая строка, а затем последовательно вместо пустых значений полей записываются пользовательские значения:

```
USE firms
APPEND BLANK
REPLACE firm WITH «Роза»
REPLACE address WITH «Тверь»
REPLACE telephone WITH 1234567
REPLACE director_f WITH «Степанов»
REPLACE account WITH 001050010
```

2.2.4. Фильтрация данных

Для выбора определенных записей таблицы, удовлетворяющих заданному условию, можно воспользоваться фильтром. Команда выглядит следующим образом:

```
SET FILTER TO <условие>.
```

Если задано правило точности (SET EXACT ON), то будут отобраны только те записи, которые **полностью** удовлетворяют заданному условию, если правило точности не задано (SET EXACT OFF, по умолчанию), то будут отобраны все записи, которые удовлетворяют условию.

Чтобы отменить фильтр, необходимо выполнить команду SET FILTER TO (без атрибутов).

На рис. 2.12 приведен результат выполнения команды SET FILTER TO для различных условий и при заданном и выключенном правиле точности.

```
Command
use goods
set exact off
set filter to good_name="Сапоги"
browse
```

Goods

Good_name	Firm	Index	Price	Colour	Amoun
Сапоги женские	Мосшуз	6	2677.0000	коричнев	2
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	2
Сапоги женские	Мосшуз	9	650.0000	черные	2

а) Правило точности выключено, результат выполнения команды `SET FILTER TO` – 3 строки

```
Command
use goods
set exact on
set filter to good_name="Сапоги"
browse
```

Goods

Good_name	Firm	Index	Price	Colour	Amount

б) Правило точности включено, в результате выполнения команды `SET FILTER TO` строк, соответствующих условия не найдено

```
Command
use goods
set exact on
set filter to good_name="Сапоги женские"
browse
```

Goods

Good_name	Firm	Index	Price	Colour	Amou
Сапоги женские	Мосшуз	6	2677.0000	коричнев	
Сапоги женские	Мосшуз	9	650.0000	черные	

в) Правило точности включено, результат выполнения команды `SET FILTER TO` – 2 строки

Рис. 2.12. Результат выполнения команды SET FILTER TO

2.3. Работа с индексами

2.3.1. Создание индексов

Индексы используются для логического упорядочивания записей и организации связей между таблицами, а также для ускорения доступа к данным и для управления порядком отображения записей таблицы [3].

Под индексным ключом понимается имя поля таблицы или выражение, включающее совокупность имен полей, по которым логически упорядочена таблица [1]. Индексы бывают простые (состоящие из одного индексного выражения) и составные (состоящие из нескольких индексных выражений). Индексные выражения могут состоять из одного или нескольких полей таблицы.

В Visual FoxPro существует четыре типа индексов (табл. 2.1.) [2, 3].

Таблица 2.1. Типы индексов

Тип	Описание
Regular	Значение индексного выражения записывается для каждой строки таблицы. При наличии одного и того же значения для нескольких строк в индексном файле будет указатель на каждую из них. При просмотре таблицы такие строки появляются в порядке их ввода
Unique	Значение индексного выражения записывается только для первой из повторяющихся строк и только на нее в индексном файле есть указатель. При просмотре таблицы видна только одна (первая) из строк с одинаковым значением индексного выражения
Candidate	Создается уникальный индекс, не содержащий полей с пустыми значениями. Он является кандидатом на роль первичного ключа, но не является таковым, так как в таблице может быть только один первичный ключ
Primary	Первичный ключ. В качестве первичного (Primary) ключа может быть выбран один индекс, удовлетворяющий требованиям индекса типа Candidate. Используется для связывания таблиц и определения условий целостности данных. Возможен только для таблиц, включенных в базу данных

Индексы таблиц хранятся в файлах с расширением .CDX (для составных индексов) или .IDX (для простых индексов).

Индексы таблицы можно задать на вкладке ***Indexes*** окна конструктора таблиц ***Table Designer*** (см. п.2.1.1.1, рис. 2.4) или с помощью командного окна.

2.3.1.1. Создание индексов через командное окно

Если индекс создается через командное окно, то необходимо выполнить команды:

```
SELECT <имя_таблицы>  && если таблица открыта
USE <имя_таблицы>  && если таблица не открыта
INDEX ON <имя_поля> TO <имя_индекса>;
  [ASCENDING/DESCENDING] [UNIQUE/CANDIDATE].
```

По умолчанию (если не указан тип индекса `unique/candidate`) создается индекс типа `Regular` (см. табл. 2.1).

Для создания составных индексов используется команда
`INDEX ON <имя_поля> TAG <имя_индекса>.`
В этом случае индексный файл будет иметь расширение .CDX.

Команда `SET INDEX TO <имя_индекса>` открывает один или несколько существующих индексных файлов, используемых с текущей таблицей [3].

Например, создадим два индекса – упорядочим записи таблицы goods по полям good_name и colour:

```
USE GOODS  && открытие таблицы
&& создание индекса по полю good_name:
INDEX ON GOOD_NAME TO IND_GOOD
&& создание индекса по полю colour:
INDEX ON COLOUR TO IND_COL
```

Просмотрим записи таблицы в порядке, соответствующем индексу ind_good:

```
SET INDEX TO IND_GOOD  && открытие индекса
BROWSE          && просмотр таблицы
```

Результат выполнения этих команд приведен на рис. 2.13,а. Теперь просмотрим записи таблицы в порядке, соответствующему индексу ind_col (рис. 2.13,б):

```
SET INDEX TO IND_COL              && открытие индекса
BROWSE          && просмотр таблицы
```

Goods

Good_name	Firm	Index	Price	Colour	Amount
Ботинки мужские	Скороход	5	2000.0000	черные	150
Калоши	Богатырь	11	203.0000	черные	500
Кеды	Мосшуз	1	100.0000	синие	100
Кеды	Скороход	2	200.0000	красные	20
Сапоги женские	Мосшуз	6	2677.0000	коричнев	250
Сапоги женские	Мосшуз	9	650.0000	черные	200
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	200
Тапочки	Богатырь	4	60.0000	розовые	2000
Туфли женские	Мосшуз	3	300.0000	черные	100
Туфли мужские	Intershoes LTD	10	1450.0000	черные	350
Шнурки	Vermishelle	8	15.0000	зеленые	3000

а) упорядочивание по полю good_name

Goods

Good_name	Firm	Index	Price	Colour	Amount
Шнурки	Vermishelle	8	15.0000	зеленые	3000
Сапоги женские	Мосшуз	6	2677.0000	коричнев	250
Кеды	Скороход	2	200.0000	красные	20
Тапочки	Богатырь	4	60.0000	розовые	2000
Кеды	Мосшуз	1	100.0000	синие	100
Туфли женские	Мосшуз	3	300.0000	черные	100
Ботинки мужские	Скороход	5	2000.0000	черные	150
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	200
Сапоги женские	Мосшуз	9	650.0000	черные	200
Туфли мужские	Intershoes LTD	10	1450.0000	черные	350
Калоши	Богатырь	11	203.0000	черные	500

б) упорядочивание по полю colour

Рис. 2.13. Результат индексирования таблицы goods

Существует еще один способ создания уникальных и неуникальных индексов.

Команда `SET UNIQUE ON/OFF` определяет, поддерживает ли файл индекса записи с повторяющимися значениями ключа индекса.

Например, в результате выполнения последовательности команд

```
SET UNIQUE ON
INDEX ON GOOD_NAME TO IND_UNIQ
```

будет создан индекс только по уникальным значениям поля good_name (рис. 2.14).

```
use goods
set uniq on
index on good_name to ind_uniq
browse
```

Goods

Good_name	Firm	Index	Price	Colour	Amount
Ботинки мужские	Скороход	5	2000.0000	черные	150
Калоши	Богатырь	11	203.0000	черные	500
Кеды	Мосшуз	1	100.0000	синие	100
Сапоги женские	Мосшуз	6	2677.0000	коричнев	250
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	200
Тапочки	Богатырь	4	60.0000	розовые	2000
Туфли женские	Мосшуз	3	300.0000	черные	100
Туфли мужские	Intershoes LTD	10	1450.0000	черные	350
Шнурки	Vermishelle	8	15.0000	зеленые	3000

Рис. 2.14. Уникальное индексирование по полю good_name

В результате выполнения последовательности команд

```
SET UNIQUE OFF
INDEX ON GOOD_NAME TO IND_NON_UNIQ
```

будет создан индекс по всем значениям поля good_name (рис. 2.15).

```
use goods
set uniq off
index on good_name to ind_non_uniq
browse
```

Goods

Good_name	Firm	Index	Price	Colour	Amount
Ботинки мужские	Скороход	5	2000.0000	черные	150
Калоши	Богатырь	11	203.0000	черные	500
Кеды	Мосшуз	1	100.0000	синие	100
Кеды	Скороход	2	200.0000	красные	20
Сапоги женские	Мосшуз	6	2677.0000	коричнев	250
Сапоги женские	Мосшуз	9	650.0000	черные	200
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	200
Тапочки	Богатырь	4	60.0000	розовые	2000
Туфли женские	Мосшуз	3	300.0000	черные	100
Туфли мужские	Intershoes LTD	10	1450.0000	черные	350
Шнурки	Vermishelle	8	15.0000	зеленые	3000

Рис. 2.15. Неуникальное индексирование по полю good_name

2.3.1.2. Создание индексов в конструкторе таблиц

Чтобы создать индексы в конструкторе таблиц, необходимо открыть конструктор таблиц командой `MODIFY STRUCTURE` и перейти к вкладке ***Indexes*** (рис. 2.16). В столбец ***Name*** вводится имя индекса, а из раскрывающегося списка столбца ***Type*** выбирается необходимый тип индекса (см. табл. 2.1). В столбце ***Expression*** вводится выражение, по которому строится индекс (в самом простом случае это имя поля (атрибута) таблицы). Для сохранения изменений необходимо нажать на кнопку ***OK***.

Просмотр записей таблицы в порядке, соответствующем индексам, аналогичен рассмотренному в п.2.3.1.1.

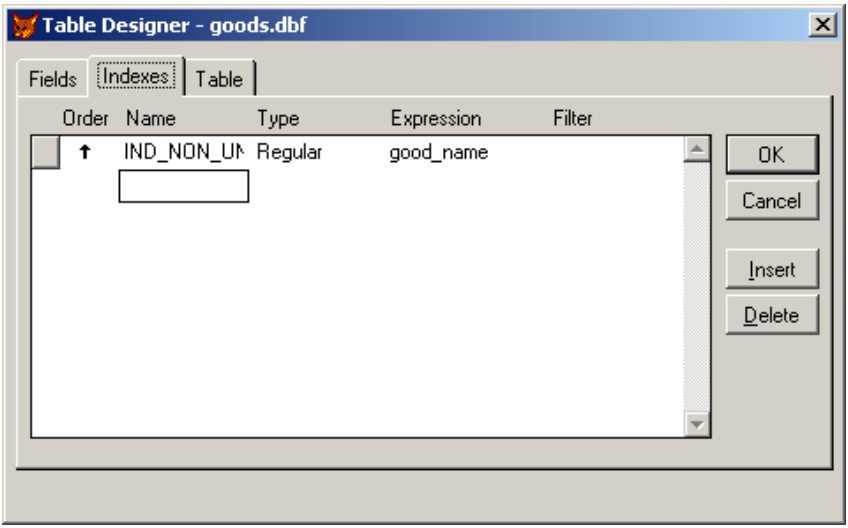


Рис. 2.16. Определение индексов в окне конструктора таблиц

2.3.2. Удаление индексов

Индексы можно удалять с помощью конструктора таблиц или с помощью команд.

Чтобы удалить индексы в конструкторе таблиц, необходимо перейти на вкладку ***Indexes***, указать индекс, подлежащий удалению, и нажать кнопку ***Delete*** (см. рис. 2.16).

Для удаления индекса из составного индекса нужно выполнить команду `DELETE TAG <имя_индекса>` в командном окне.

2.4. Реструктуризация базы данных

2.4.1. Создание базы данных

Использование базы данных значительно упрощает работу с данными, так как в ней реализованы мощные и полезные функции, используемые при работе с таблицами. Например, в файле с расширением DBF нельзя дать полю имя, содержащее более 10 символов, но если таблица включена в базу данных, то имя поля таблицы может содержать до 128 символов. Никакие правила (RULE), значения по умолчанию (DEFAULT) и триггеры попросту невозможны в файле DBF вне файла базы данных, точнее невозможно их автоматическое выполнение [4].

Использование файла базы данных позволяет выполнять операции, которые крайне сложно организовать другими способами. Например, такая операция как «транзакция» может быть реализована только среди таблиц, включенных в базу данных. В принципе, этот процесс можно организовать и со свободными таблицами, но это потребует от программиста значительных усилий [4].

Поэтому в большинстве случаев целесообразно создавать базу данных, включающую и таблицы, и связи между ними.

Базы данных, как и таблицы, можно создать двумя способами: через меню среды Visual FoxPro и через командное окно.

2.4.1.1. Создание базы данных через меню

Если база данных создается через меню, то следует выбрать пункт меню ***File→New*** и в открывшемся диалоговом окне ***New*** (см. рис. 2.2) выбрать переключатель ***Database*** и нажать кнопку ***New*** File.

На экране появится диалоговое окно ***Create*** (аналогично рис. 2.9). В нем следует выбрать папку (если предварительно не была установлена папка по умолчанию, например, с помощью команды `SET DEFAULT TO <имя_папки>`, то открывается папка, в которой установлен пакет Visual FoxPro) и в поле ***Enter database*** указать имя файла базы данных. При создании базы данных следует присваивать ей имя, отражающее существо хранимой информации, но в рамках правил операционной системы Windows [2, 4].

Для нашего примера создадим базу данных shop. После сохранения имени базы данных откроется окно ***Database Designer*** – конструктор базы данных (рис. 2.17). Одновременно может быть открыто несколько окон конструктора базы данных для каждой используемой базы данных.

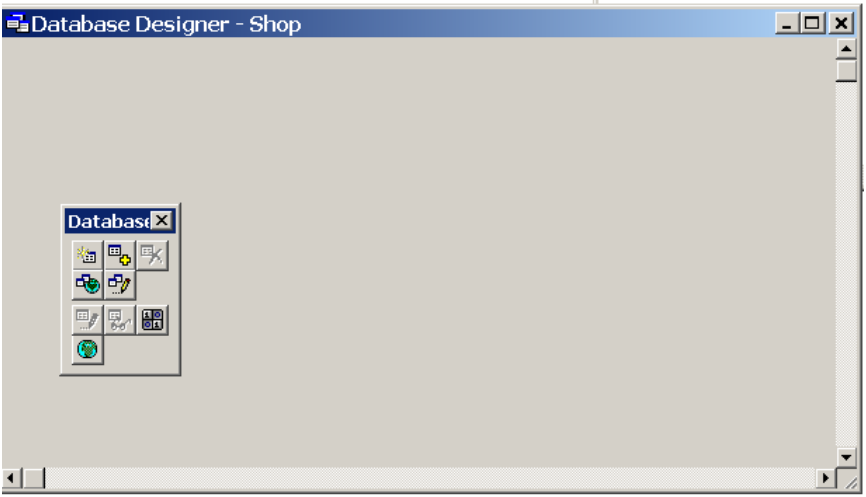


Рис. 2.17. Окно конструктора базы данных

Набор инструментария ***Database Controls***, содержит следующие кнопки (слева направо) [3]:

New table – создание новой таблицы базы данных;
Add table – добавление в базу данных свободной таблицы;
Remove table – удаление таблицы из базы данных;
New remote view – создание нового удаленного вида;
New local view – создание нового локального вида;
Modify table – изменение структуры таблицы;
Browse table – просмотр содержимого таблицы;
Edit stored procedures – вызов окна редактирования текстов хранимых процедур;
Connections – вызов окна определения соединения.

Чтобы открыть конструктор базы данных для уже созданной базы данных, необходимо выполнить команду

```
MODIFY DATABASE <имя_базы_данных>.
```

2.4.1.2. Создание базы данных через командное окно

Если база данных задается с помощью командного окна (см. рис. 2.1), то необходимо выполнить команду `CREATE DATABASE`. На экране появится диалоговое окно ***Create*** (аналогично рис. 2.9). При выполнении команды

```
CREATE DATABASE <имя_базы_данных>
```

база данных сохранится на диск в текущую папку и сразу будет открыто окно конструктора базы данных.

2.4.2. Работа с таблицами в базе данных

2.4.2.1. Добавление таблиц в базу данных

Если таблицы созданы раньше базы данных, то их можно включить в базу данных после ее открытия либо командой `ADD TABLE`, либо с помощью мыши в окне конструктора базы данных ***Database Designer*** (по нажатию правой кнопки мыши в выпадающем меню выбрать пункт меню ***Add Table*** или нажать на кнопку соответствующего инструмента ***Database Control***). В открывшемся окне необходимо выбрать файл таблицы, которую необходимо включить в базу данных (рис. 2.18).

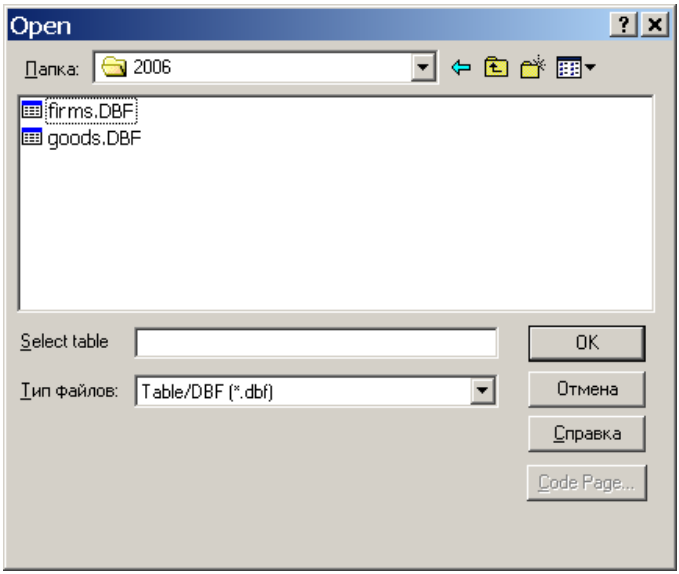


Рис. 2.18. Выбор таблицы

Если база данных создана раньше, чем таблицы, то создать таблицы в базе данных можно либо с помощью мыши в окне конструктора базы данных ***Database Designer*** (по нажатию правой кнопки мыши в выпадающем меню выбрать пункт меню ***New Table***), либо при открытой базе данных создать таблицу, как описано в п.2.1.1.

Изменить структуру базы данных можно с помощью команды `MODIFY DATABASE`. После ее ввода будет открыто окно конструктора базы данных (см. рис. 2.17), в котором можно сделать необходимые изменения.

Для нашего примера включим в базу данных shop ранее созданные таблицы firms и goods (рис. 2.19).

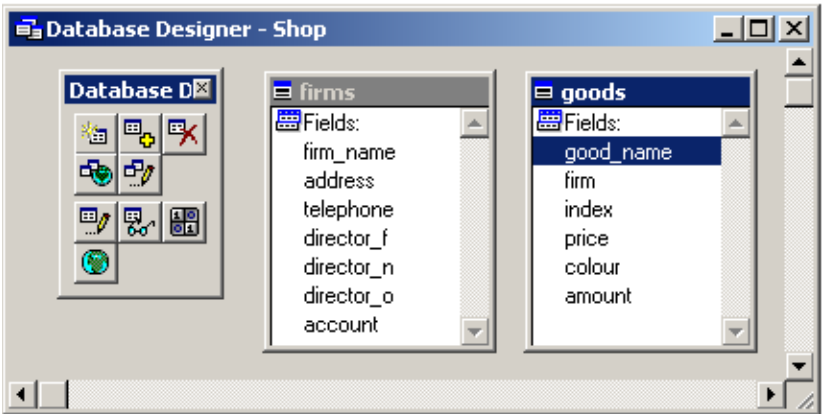


Рис. 2.19. Окно конструктора базы данных с добавленными таблицами firms и goods

2.4.2.2. Редактирование таблиц в базе данных

Редактирование структуры таблиц, включенных в базу данных, осуществляется аналогично редактированию свободных, т.е. не включенных в базу данных таблиц (см. п.2.1.2).

Для вызова конструктора таблиц необходимо либо нажать на кнопку соответствующего элемента панели инструментов ***Database Controls***, либо щелкнуть правой кнопкой мыши на таблице внутри конструктора базы данных и выбрать пункт меню ***Modify***.

2.4.2.3. Удаление таблиц из базы данных

Таблицы можно удалить из базы данных либо через командное окно, либо через меню; причем таблицу можно либо только исключить из базы данных, либо удалить с диска.

Для того чтобы исключить таблицу из базы данных (**но не удалить ее с диска**), в командном окне необходимо выполнить команду `REMOVE TABLE <имя_таблицы>`.

Если необходимо не только исключить таблицу из базы данных, но и удалить ее с диска, то необходимо в командном окне выполнить команду `REMOVE TABLE <имя_таблицы> DELETE`.

При исключении таблицы из базы данных через меню можно воспользоваться кнопкой исключения таблиц в панели инструментов ***Database Controls*** или щелкнуть правой кнопкой мыши на таблице внутри конструктора базы данных и выбрать пункт меню **Delete**. Visual FoxPro откроет окно, показанное на рис. 2.20. При нажатии на кнопку ***Remove*** таблица будет исключена из базы данных, при нажатии на кнопку ***Delete*** – удалена с диска. Кнопка ***Cancel*** – отмена операции.

Рис. 2.20. Подтверждение удаления таблицы

2.4.3. Связи в базе данных

Структура реляционной базы данных всегда разрабатывается таким образом, чтобы каждая таблица, которая в ней находится, не содержала избыточной информации. Например, в базе данных магазина необходимо хранить данные о товарах, которые продаются в данном магазине. Как следствие, нужно также каким-то образом хранить и названия фирм, которые производит эти товары. Если для этих целей будет использоваться одна таблица, то станет очевидным нерациональное использование памяти компьютера, так как для каждого товара придется хранить в соответствующей запи-

си названия и другие атрибуты фирм, которые будут повторяться много раз. Поэтому необходимо создать две таблицы, которые будут между собой взаимосвязаны. При этом для созданных таблиц необходимо установить связи, чтобы, например, по названию фирмы в таблице фирм определить все записи с товарами этой фирмы из таблицы товаров [5]. Связи бывают «один-к-одному» (1:1), «один-ко-многим» (1:M) и «многие-ко-многим» (M:N) (последние в реляционных СУБД не поддерживаются). Связи между таблицами в базе данных используются при формировании запросов, создании отчетов или разработке форм. Создать связь между таблицами можно только после создания индексов. В родительской таблице должен быть создан индекс типа Primary или Candidate.

Связи могут быть временными (только для текущего сеанса работы с базой данных) и постоянными (сохраняются и остаются на следующий сеанс работы).

2.4.3.1. Создание связей

Временная связь создается командой `SET RELATION` с большим числом дополнительных параметров.

Постоянная связь создается в окне конструктора базы данных. Поля, по которым будут связываться таблицы, должны иметь одинаковый тип данных, но названия их не обязательно должны совпадать. Для нашего примера это поле firm_name таблицы goods и поле firm таблицы firms. Для создания связи необходимо щелкнуть мышью на имени индекса (первичного ключа) родительской таблицы и, не отпуская клавишу мыши, «перетащить» его к индексу в дочерней таблице.

Установим отношение по полям таблиц firms и goods, содержащим название фирмы. Для этого щелкнем левой кнопкой мыши на индексе (первичном ключе) firm в таблице firms и перетащим его к индексу firm_name в таблице goods (рис. 2.21). Появившаяся линия означает, что связь создалась.

Линия связи, оканчивающаяся тремя штрихами (⊣≪) показывает, что отношение к данной таблице – «многие». Линия связи, оканчивающаяся одним штрихом (⊣—) показывает, что отношение к таблице – «один».

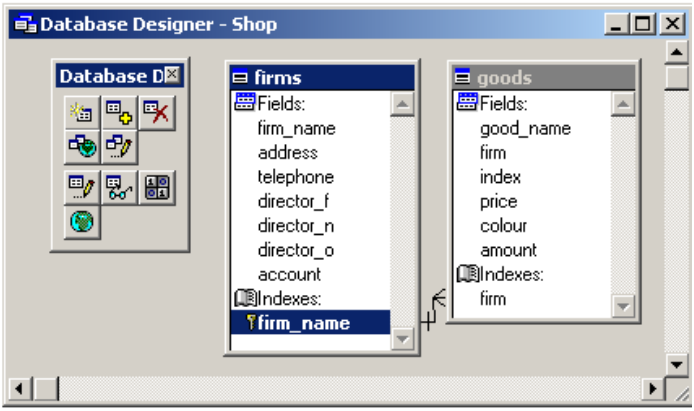


Рис. 2.21. Установка связи между таблицами

2.4.3.2. Редактирование связей

Для редактирования связи можно выполнить двойной щелчок мышью на линии связи между таблицами и в появившемся меню выбрать пункт меню ***Edit Relationship***. Будет открыто окно редактирования связи, в котором для каждой таблицы в выпадающем списке под ее именем указаны ее индексы, и необходимо выбрать и указать индексы, по которым осуществляется связь (рис. 2.22).

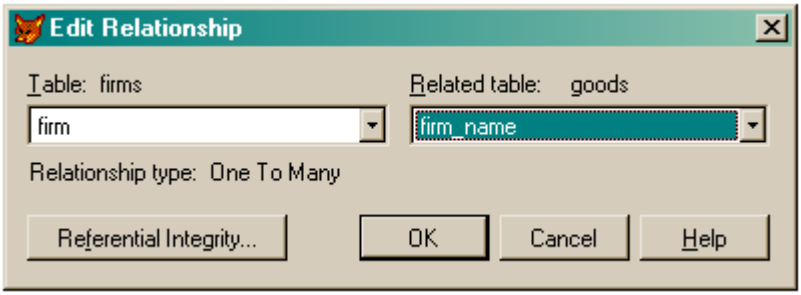


Рис. 2.22. Редактирование связи

2.4.3.3. Удаление связей

Для удаления связи можно выполнить двойной щелчок мышью на линии связи между таблицами и в появившемся меню выбрать опцию ***Remove Relationship*** или выделить связь и нажать клавишу ***Delete***.

2.4.4. Удаление базы данных

Удаление базы данных можно осуществить с помощью команды `DELETE DATABASE <имя_базы_данных> DELETETABLES`. При выполнении этой команды будет удалена и база данных, и все включенные в нее таблицы.

Команда `DELETE DATABASE` без фразы `DELETETABLES` не удаляет одновременно с базой данных все включенные в нее таблицы, а только удаляет в этих таблицах связь с базой данных и оставляет их на диске как свободные таблицы.

Заключение

В ходе лабораторной работы студенты должны научиться создавать таблицы и базу данных, работать с разными рабочими областями, устанавливать связь между таблицами, добавлять/удалять записи из таблицы, модифицировать таблицы и базу данных, индексировать таблицы, осуществлять отбор записей по фильтру.

Вопросы для самоконтроля

1. Какая команда создает таблицу?
2. Какие Вы знаете числовые типы данных?
3. Как можно изменить структуру таблицы (например, изменить имя поля или тип данных)?
4. Что такое рабочая область и для чего она используется?
5. Как просмотреть заполненную таблицу (введенные данные)?
6. Как добавить строчку данных в существующую таблицу?
7. Как изменить данные в таблице?
8. Как упорядочить записи таблицы для просмотра?
9. В чем различие понятий уникальный и неуникальный индекс?
10. Чем отличаются временная и постоянная связи?
11. Что происходит при удалении таблицы из базы данных?
12. Как выбрать определенные записи из таблицы?

Практическое занятие 3
Программирование на языке FoxPro

Цель работы: научиться создавать программы в среде Visual FoxPro.

Visual FoxPro позволяет сохранять последовательность команд (программу) в командном файле и впоследствии выполнять их. Файл с текстом программы имеет расширение .PRG.

3.1. Создание программ

Чтобы начать писать новую программу, можно выбрать пункт меню ***File→New***, в диалоговом окне ***New*** выбрать переключатель ***Programm*** (см. рис. 2.2) и нажать на кнопку ***New*** или в командном окне ввести команду `MODIFY COMMAND`. В результате этих операция будет открыто окно редактирования программы (рис. 3.1).

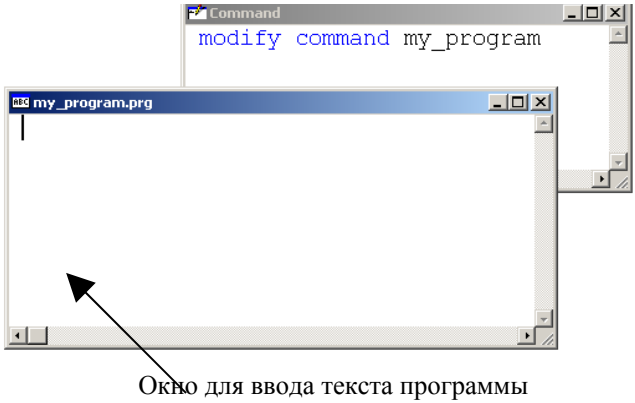


Рис. 3.1. Создание программ в среде Visual FoxPro

Программа состоит из последовательности команд, которые пишутся построчно. Символ «*» в начале строки и символ «&&» в произвольной позиции строки означают начало комментария (т.е. текст, следующий за этими символами, не интерпретируется FoxPro как команда). Символ «;» – признак переноса продолжения команды на следующую строку.

При сохранении и выходе из окна редактирования программы необходимо присвоить программе соответствующее имя.

3.2. Редактирование программ

Для редактирования программы необходимо выполнить команду `MODIFY COMMAND <имя_программы>` с указанием имени программы, которую необходимо отредактировать. Откроется окно редактирования программы, аналогичное рис. 3.1. После внесения изменений необходимо сохранить файл для сохранения внесенных изменений.

3.3. Удаление программ

Удалить файл программы с диска можно с помощью команды `DELETE FILE <имя_программы>.prg`.

3.4. Выполнение программ

Выполнить сохраненную в файле программу можно, запустив ее через пункт меню ***Program→Run*** или из командного окна командой `DO <имя_программы>`.

3.5. Примеры программ

Пример 1. Программа индексирования данных таблицы

Напишем программу, которая упорядочивает строки таблицы goods по выбранному полю и выдает результат индексирования на экран. Входными данными является имя поля, по которому пользователь хочет осуществить индексирование, выходными данными является упорядоченная по заданному полю таблица.

В командном окне введем команду

```
MODIFY COMMAND my_sort_program
```

и в открывшемся окне введем следующие строки:

```
CLEAR                                   && очистка экрана
CLOSE DATA              && закрыть все открытые таблицы
USE goods                          && открыть таблицу Goods
BROWSE                              && просмотреть таблицу Goods
* переменной shablon присвоить значение « »
shablon=« »
* выводим приглашение для ввода значения переменной shablon
@5,1 SAY «Введите имя поля для индексирования»;
 GET shablon
* и считываем значение переменной
READ
* проиндексируем (упорядочиваем) таблицу по полю, указанному в
* переменной shablon
INDEX ON &shablon TO ind
BROWSE                               && просмотреть содержимое таблицы
CLOSE DATA              && закрыть все открытые таблицы
```

Теперь закроем программное окно кнопкой  , ответив утвердительно на запрос о сохранении изменений в тексте программы, и запустим программу, например, из командного окна:

```
DO my_sort_program.
```

Результат работы программы приведен на рис. 3.2. В случае коррекного ввода названия имени поля, по которому необходимо упорядочить таблицу на экран будет выведена упорядоченная (проиндексированнная) таблица. Для нашего примера данные таблицы Goods упорядочены по полю price.

Good_name	Firm	Index	Price	Colour	Amount
Шнурки	Vermishelle	8	15.0000	зеленые	3000
Тапочки	Богатырь	4	60.0000	розовые	2000
Кеды	Мосшуз	1	100.0000	синие	100
Кеды	Скороход	2	200.0000	красные	20
Калоши	Богатырь	11	203.0000	черные	500
Туфли женские	Мосшуз	3	300.0000	черные	100
Сапоги женские	Мосшуз	9	650.0000	черные	200
Туфли мужские	Intershoes LTD	10	1450.0000	черные	350
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	200
Ботинки мужские	Скороход	5	2000.0000	черные	150
Сапоги женские	Мосшуз	6	2677.0000	коричнев	250

Рис. 3.2. Результат выполнения программы сортировки

В случае некорректного ввода данных на экран будет выведено сообщение об ошибке.

Пример 2. Поиск данных в индексированной таблице

Напишем программу, предлагающую ввести имя таблицы, имя поля этой таблицы и значение атрибута для поиска. Результатом поиска является строка, содержащая указанное значение атрибута.

Примерный текст программы поиска приведен ниже.

```
clear
close all
*Задаем имя таблицы в переменной ttt
ttt=« »
@3,1 say «Введите имя таблицы» get ttt
read
use &ttt
*Задаем имя поля в переменной yyy
yyy=« »
@5,1 say «Введите имя поля для поиска» get yyy
read
*Индексируем таблицу по этому полю
index on &yyy to xx
*Считываем значение атрибута в переменную shablon
shablon=« »
@6,1 say «Введите шаблон для поиска» get shablon
read
*Ищем значение атрибута. Функция alltrim отсекает пробелы слева и
*справа от значения
seek alltrim(shablon)
*Показываем содержимое таблицы
browse
```

При условии корректного ввода данных в результате выполнения программы поиска на экран выводится содержимое выбранной таблицы, причем курсор установлен на строке, содержащей заданное значение атрибута. Результат работы программы приведен на рис. 3.3.

В случае некорректного ввода данных на экран будет выведено сообщение об ошибке.

Введите имя таблицы goods

Введите имя поля для поиска firm
Введите шаблон для поиска Мосшуз

Goods

Good_name	Firm	Index	Price	Colour	Amount
Сапоги мужские	Intershoes LTD	7	1999.0000	черные	200
Туфли мужские	Intershoes LTD	10	1450.0000	черные	350
Шнурки	Vermishelle	8	15.0000	зеленые	3000
Тапочки	Богатырь	4	60.0000	розовые	2000
Калоши	Богатырь	11	203.0000	черные	500
Кеды	Мосшуз	1	100.0000	синие	100
Туфли женские	Мосшуз	3	300.0000	черные	100
Сапоги женские	Мосшуз	6	2677.0000	коричнев	250
Сапоги женские	Мосшуз	9	650.0000	черные	200
Кеды	Скороход	2	200.0000	красные	20
Ботинки мужские	Скороход	5	2000.0000	черные	150

Рис. 3.3. Результат выполнения программы поиска

Пример 3. Построение графиков

Построим график функции $y = x^2$, для чего в цикле рассчитаем значение функции *y* для девяти значений аргумента *x* (от 1 до 9):

```
x=1
y=0
do while x<10
 y=x*x
 @x,y say «*(»+str(x,2)+ «,»+str(y,2)+ «)»
 x=x+1
enddo
```

Результат выполнения программы приведен на рис. 3.4. Обратите внимание, что график получился «перевернутым», то есть в конструкции `@x,y x` – это номер строки экрана (строки нумеруются сверху вниз), а `y` – это номер столбца (нумерация слева направо).

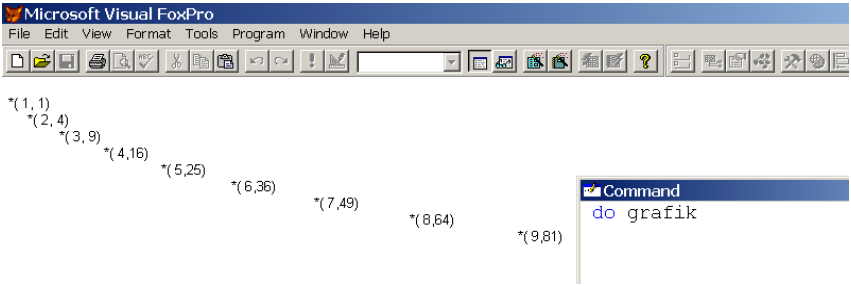


Рис. 3.4. Результат работы программы grafik

Пример 4. Рисование линий

Программа osi рисует оси координат и подписывает их (отрезок прямой изображается на экране командой `@x1,y1 to x2,y2`, где `x1,y1` – координаты начала, а `x2,y2` – координаты конца отрезка). Текст программы:

```
@5,5 to 25,5
@25,5 to 25,100
@5,7 say «Ордината»
@26,90 say «Абсцисса»
```

Результат выполнения программы приведен на рис. 3.5.

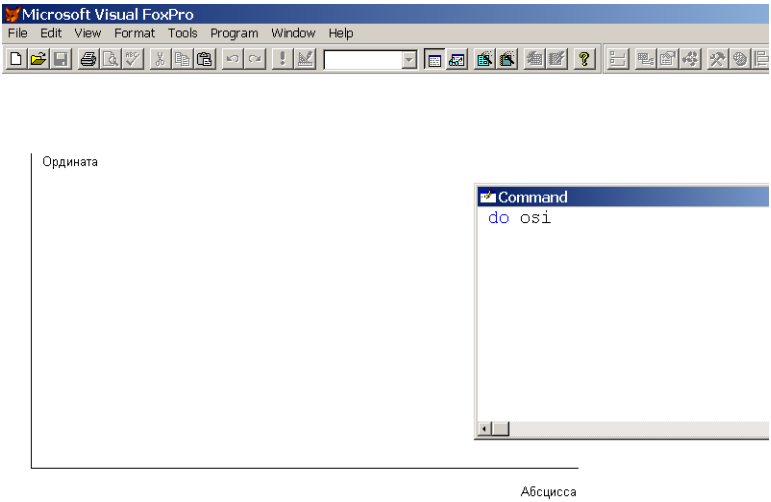


Рис. 3.5. Результат выполнения программы osi

Пример 5. Подсчет количества строк таблицы, удовлетворяющих некоторому условию

Программа calculate подсчитывает количество товаров заданного поставщика (фирмы) в базе данных.

На экран выдается содержимое поля good_name таблицы goods и программа просит пользователя выбрать наименование товара, после чего название фирмы-производителя сохраняется в переменной choice и программа подчитывает количество строк, удовлетворяющих условию `firm=choice` (название фирмы-производителя совпадает с заданным в переменной choice значением) в таблице goods, а в таблице firms находит информацию о фирме-производителе выбранного товара и выводит ее на экран.

```
clear
select 1
use goods
@2,2 say «ВЫБЕРИТЕ ТОВАР И НАЖМИТЕ ESC»
browse fields good_name
@4,2 say «ВЫ ВЫБРАЛИ »+good_name+« ОТ ФИРМЫ »+firm
choice=firm
select 2
use firms
index on firm_name to ind_firm
seek choice
@6,2 say «АДРЕС ПОСТАВЩИКА »+address
@7,2 say «ТЕЛЕФОН »+str(telephone,14)
select 1
count for firm=choice to number_goods
@9,2 say «ВСЕГО ТОВАРОВ ЭТОГО ПОСТАВЩИКА »+str(num-
ber_goods,2)
```

Результат выполнения программы приведен на рис. 3.6.

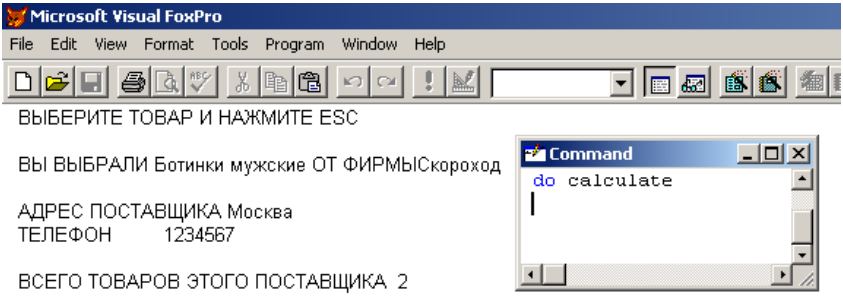


Рис. 3.6. Результат выполнения программы calculate

Пример 6. Программа построения гистограммы

Напишем программу, которая строит гистограмму. В зависимости от задания, гистограмма может быть построена по данным одной или двух таблиц.

Рассмотрим оба случая: построим гистограмму, показывающую количество товаров в магазине из различных городов (адресов) по всем значениям поля таблицы firms, и гистограмму, показывающую количество товаров различных фирм в магазине по заданному пользователем количеству и значениям поля таблицы firms.

Приведем текст обеих программ.

Для построения первой гистограммы требуется информация из двух таблиц: из таблицы firms берутся названия городов (адресов) фирм, и по таблице goods подсчитывается количество наименований товаров. Текст программы построения гистограммы по первому способу (подсчитывается количество товаров по каждому городу, используется две таблицы):

```
* программа «Гистограмма 1»
clear
close all
num_col=5 && количество столбцов гистограммы
select 1
use firms && в раб.области 1 открываем таблицу firms
set uniq off && неуникальная сортировка
index on address to adr_non_uniq && по полю address
set uniq on && уникальная сортировка
```

```
index on address to adr_uniq && по полю address
count for address!=« » to num_col
&&подсчет столбиков
go top && переход на первую строку таблицы firms
select 2
use goods && в раб.области 2 открываем таблицу goods
@10,2 say «Количество товаров из городов»
@2,1 to 20,1 && вертикальная ось
@20,1 to 20,70 && горизонтальная ось
i=1 && i – переменная цикла
do while i<=num_col && цикл по столбикам (А)
 select 1
 set index to adr_uniq
 brow fields address
 && показать все уникальные адреса для выбора курсором
 choice=address && запоминаем адрес
 x=0
 set index to adr_non_uniq
 && неуникальное индексирование
 set filter to address=choice
 && выбираем фирмы с нужным адресом
 go top              && начиная с первой записи
 do while !EOF()
 && цикл по фирмам с однаковыми адресами –
пока не кончится файл (Б)
 y=0 && товаров у конкретной фирмы
 firma=firm_name && запоминаем название фирмы
 select 2
 count for firm=firma to y
 && считаем количество товаров этой фирмы
 x=x+y
 && увеличиваем количество товаров ВСЕГО
 select 1
 if !EOF()
 && если еще не дошли до конца файла
 skip 1
 && переходим на следующую запись
 endif
 enddo
```

```
&& конец цикла Б (по фирмам с одинаковыми адресами)
 @20-x,10*(i-1)+5 to 20,i*10 && рисуем столбик
 @20+i,10*(i-1)+5 say choice
 && подписываем адрес (город)
 @20-x-1,10*(i-1)+6 say str(x,5,0)
 && пишем число
 i=i+1 && увеличиваем переменную цикла
 set filter to
enddo && конец цикла А (по различным адресам)
```

Результат работы программы «Гистограмма 1» приведен на рис. 3.7.

Рис. 3.7. Результат работы программы построения гистограммы по двум таблицам

Для построения гистограммы вторым способом требуется информация только из одной таблицы goods, так как она содержит и названия фирм, и названия товаров. Однако в отличие от первого способа, гистограмма строится не по всем различным фирмам, а по заданному пользователем количеству и названиям фирм.

Текст программы построения гистограммы по заданному пользователем количеству и значениям поля таблицы (подсчитывается количество товаров по каждой фирме, используется одна таблица):

```
*программа «Гистограмма 2»
clear &&очистить экран
close data && закрываем все открытые таблицы
num_col=5
&&значение количества столбцов гистограммы по умолчанию
@ 1,1 say «Введите количество фирм» get num_col
&& задаем количество столбцов гистограммы
read &&считываем введенное значение
@ 20,3 to 20,100 &&рисуем ось абсцисс
@ 1,3 to 20,3 &&рисуем ось ординат
@ 10,4 say «Количество товаров различных фирм»
&&пишем название гистогрммы
i=1 &&определяем переменную счетчика цикла и
* приравниваем значение счетчика к 1
use goods &&обращаемся к таблице
do while i<=num_col &&начинаем выполнение цикла do
* от 1 до необходимого количества столбцов (num_col)
set unique on &&включаем уникальность
index on firm to f_ind
* индексируем по полю названия фирм
browse field firm
* выводим на экран только поле с названиями фирм
choice=firm &&вводим доп. переменную, в которую
* записываем выбранное значение
set unique off &&выключаем уникальность в таблице
index on firm to f_ind
count for firm=choice to kol_firm
*  подсчитываем  количество строк в таблице, удовл. условию
*  (количество товаров указанной фирмы)
@ 20-kol_firm,i*10-5 to 20,i*10
* рисуем прямоугольник с подсчитанной высотой
@20+i,10*(i-1)+5 say choice
* внизу прямоугольника подписываем название фирмы,
* по которому проводился подсчет
@ 20-kol_firm-2,i*10-3 say str(kol_firm,2)
* сверху подписываем, сколько товаров фирмы есть в БД
i=i+1 && увеличиваем значение счетчика на единицу и переходим
к следующей фирме
enddo && окончание цикла
```

Результат работы программы «Гистограмма 2» приведен на рис. 3.8.

Рис. 3.8. Результат работы программы гистограммы по одной таблице

Заключение

В ходе лабораторной работы студенты должны научиться создавать программы в среде Visual FoxPro, вносить в них изменения и запускать на выполнение.

Вопросы для самоконтроля

1. Какое расширение имеет файл, содержащий текст программы? Как создать такой файл?
2. Как запустить программу на выполнение?
3. Как внести изменения в текст существующей программы?
4. Как организовать цикл внутри программы?

Практическое занятие 4
Создание экранных форм, отчетов, этикеток и меню. Визуальное программирование

Цель работы: научиться работать с формами, отчетами и этикетками в среде Visual FoxPro – создавать, модифицировать, запускать.

Visual FoxPro предоставляет два способа создания нового файла для большинства типов файлов, включая формы, отчеты и этикетки:

- создать новый файл для проектирования объекта в среде визуального программирования;
- вызвать мастер файла (Wizard), облегчающий создание простых приложений.

При создании нового файла формы (отчета, этикетки) необходимо вручную размещать поля источника данных в окне формы (отчета, этикетки).

При вызове мастера необходимо ответить на его вопросы в последовательности диалоговых окон. Форма (отчет, этикетка) создастся автоматически [1–3,6].

4.1. Работа с формами

Формы используются для ввода и редактирования данных в таблицах. Формы предоставляют пользователю удобный интерфейс для доступа к хранимым данным с возможностью отображения их в требуемом виде.

4.1.1. Создание формы

В Visual FoxPro существует возможность создавать формы как на основании данных из одной таблицы, так и на основании данных из нескольких таблиц.

При создании форм через мастер форм Visual FoxPro предлагает два типа формы: ***Form Wizard*** (форма на основе одной таблицы) или ***One-to-Many Form Wizard*** (форма на основе двух связанных таблиц).

4.1.1.1. Создание форм по одной таблице с помощью мастера

Рассмотрим по шагам проектирование формы по таблице goods.

Для создания новой формы необходимо выбрать пункт меню ***File→New*** в главном меню. В окне создания нового объекта (см. рис. 2.2) выбираем опцию создания формы (***Form***) и нажимаем на кнопку мастера форм (***Wizard***). В появившемся окне ***Wizard Selection*** (выбор мастера) (рис. 4.1) выбираем тип создаваемой формы. В нашем случае это однотабличный тип (***Form wizard***).

Рис. 4.1. Выбор мастера форм

После нажатия кнопки ***OK*** запускается мастер проектирования формы. Мастер проектирования форм предлагает последовательность диалоговых окон. Переход от одного окна к другому происходит после нажатия кнопки ***Next*** (переход к следующему окну) или ***Back*** (возврат к предыдущему окну).

В первом окне мастера необходимо указать таблицу, для которой создается форма, и выбрать поля этой таблицы, размещаемые на форме (рис. 4.2). В области ***Databases and tables*** расположены два списка. Верхний список содержит список открытых баз данных, нижний – список таблиц выбранной базы данных или свободных таблиц.

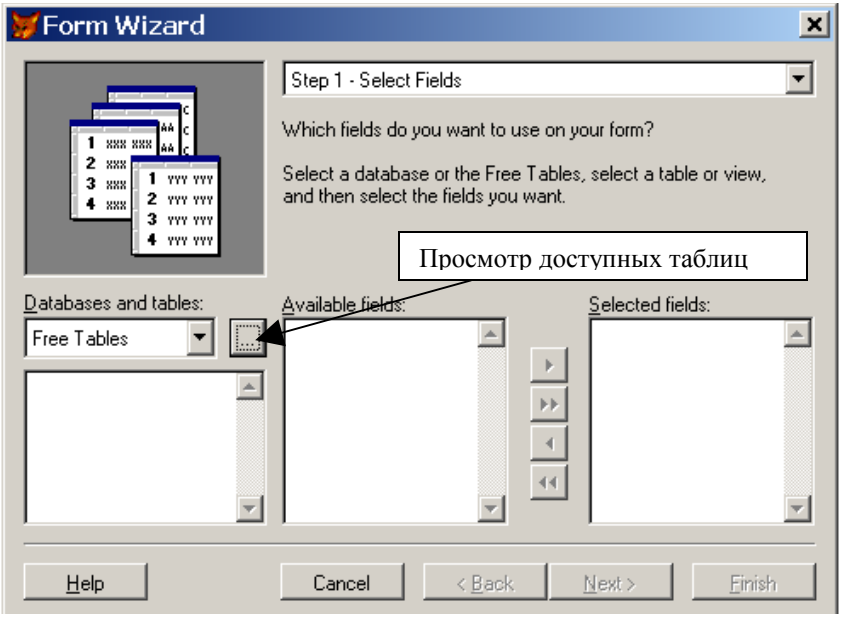


Рис. 4.2. Выбор таблицы при создании формы

Из нижнего столбца области ***Databases and tables*** мы выбираем таблицу goods, по которой и будем создавать форму. После выбора таблицы в списке ***Available fields*** появится перечень всех полей таблицы (рис. 4.3).

Из данного списка с помощью кнопок в список ***Selected fields*** переносим поля, которые хотим разместить на создаваемой форме.

Кнопки  – перемещение одного элемента списка.

Кнопки  – перемещение всего списка.

Выбрав поля, которые будут отображаться на форме, нажимаем кнопку ***Next*** для перехода к следующему шагу.

На втором шаге мастера создания форм предлагается выбрать стиль оформления формы (***Style***) и определить вид кнопок (***Button type***) (рис. 4.4).

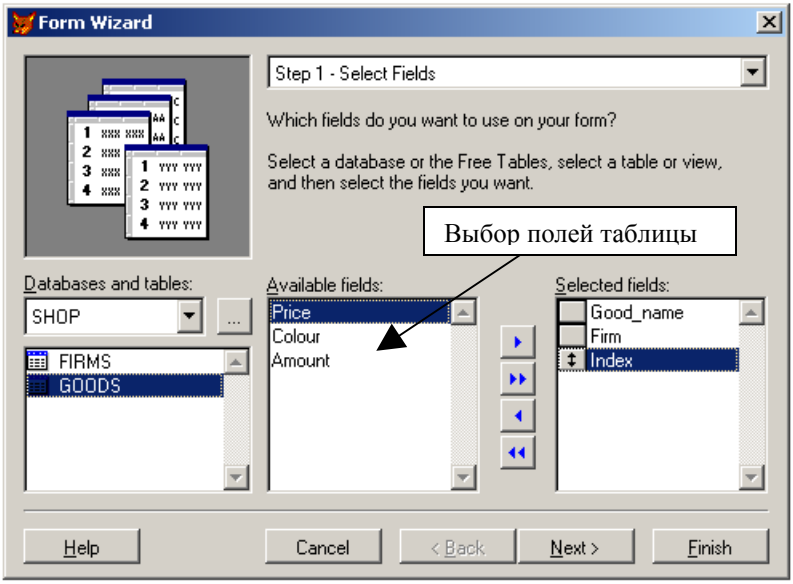


Рис. 4.3.Выбор полей таблицы для отображения в форме

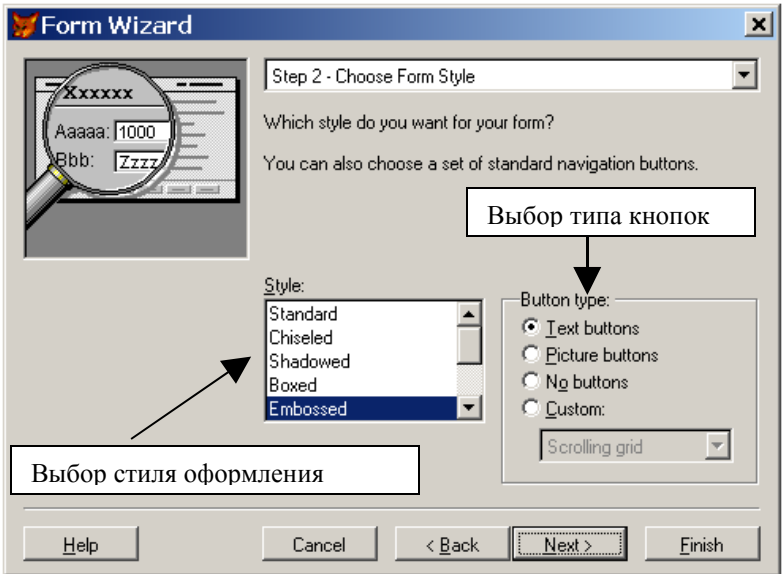


Рис. 4.4. Выбор стиля формы

В появившемся диалоговом окне мастера устанавливаем понравившийся нам стиль отображения формы и вид кнопок. Нажимаем кнопку ***Next*** для перехода к следующему шагу.

На третьем шаге можно определить порядок просмотра записей таблицы в форме, т.е. задать критерии сортировки данных, отображаемых на форме. В списке ***Available fields or index tag*** отображаются все возможные поля таблицы goods и с помощью кнопки ***Add*** их можно переместить в колонку ***Selected fields*** (рис. 4.5). С помощью кнопки ***Remove*** выбранные поля можно удалить из списка полей для сортировки данных.

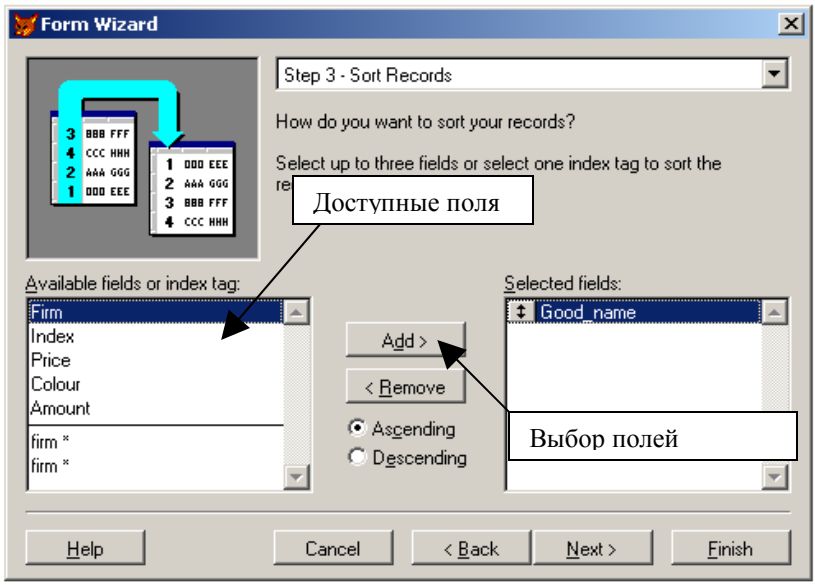


Рис. 4.5. Выбор порядка просмотра записей таблицы в форме

По полям, указанным в колонке ***Selected fields*** будет проводиться сортировка данных в форме. Нажимаем кнопку ***Next*** для перехода к следующему шагу.

На четвертом шаге создания формы с помощью мастера задаем заголовок формы «*Товары*» в поле ***Type a title for your form***, просматриваем предварительный результат (кнопка ***Preview***) , а также указываем действие с созданной формой, выбрав одну из опций (рис. 4.6):

- ***Save form for later use*** – сохранение формы;
- ***Save and run form*** – сохранение формы и ее запуск;
- ***Save form and modify it in the form designer*** – сохранение формы и открытие конструктора форм для модификации формы.

При нажатии на кнопку ***Finish*** мастер создания форм будет закрыт и выполнено указанное пользователем действие с формой.

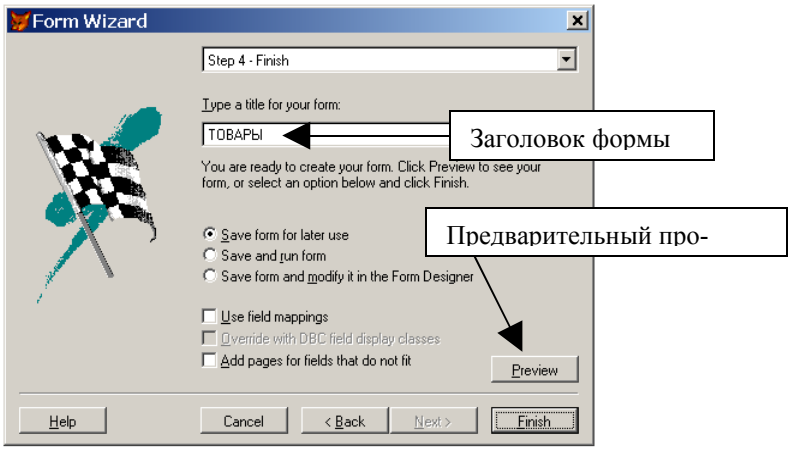


Рис. 4.6. Завершение проектирования формы

Сформированная мастером форм форма является законченным и отлаженным продуктом. При необходимости эту форму можно модифицировать таким же образом, как и форму, созданную непосредственно из нового файла, с помощью конструктора форм.

4.1.1.2. Создание форм по нескольким таблицам с помощью мастера

Рассмотрим создание формы на основе двух таблиц.

Форма «один-ко-многим» (***One-to-Many Form***) позволяет одновременно работать с данными двух логически связанных между собой таблиц.

После выбора пункта меню ***File→New*** в диалоговом окне ***New*** (см. рис. 2.2) выбираем опцию создания формы (***Form***) и нажимаем на кнопку мастера форм (***Wizard***). В появившемся диалоговом окне ***Wizard Selection*** (выбор мастера) (см. рис. 4.1) выбираем тип создаваемой формы ***One-to-Many Form Wizard***.

После нажатия кнопки ***OK*** появляется первое диалоговое окно мастера (рис. 4.7), в котором необходимо указать родительскую таблицу для создаваемой формы и выбрать поля этой таблицы, размещаемые в форме. На данном шаге мы выбираем родительскую таблицу firms. В список ***Selected fields*** переносим поля родительской таблицы, которые хотим разместить на создаваемой форме.

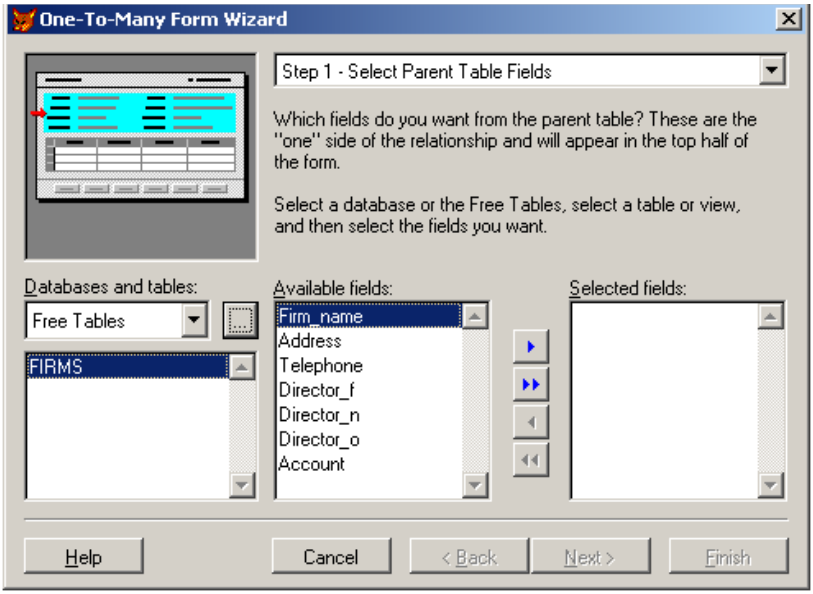


Рис. 4.7. Выбор полей родительской таблицы для отображения в форме

На следующем шаге (рис. 4.8) выбираем дочернюю таблицу (goods) и поля этой таблицы, которые будут размещены на форме.

На третьем шаге (рис. 4.9) проверяем поля, по которым устанавливается связь между таблицами. Эти поля должны иметь одинаковый тип данных, но названия их не обязательно должны совпадать. Для нашего примера это поле firm_name таблицы goods и поле firm таблицы firms (те поля, по которым строилась связь в конструкторе базы данных, см. п.2.4.3.1).

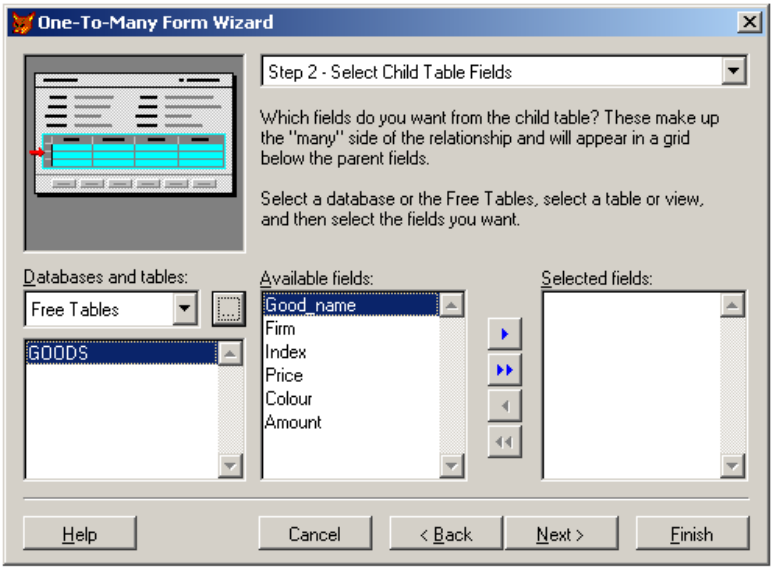


Рис. 4.8. Выбор полей дочерней таблицы для отображения в форме

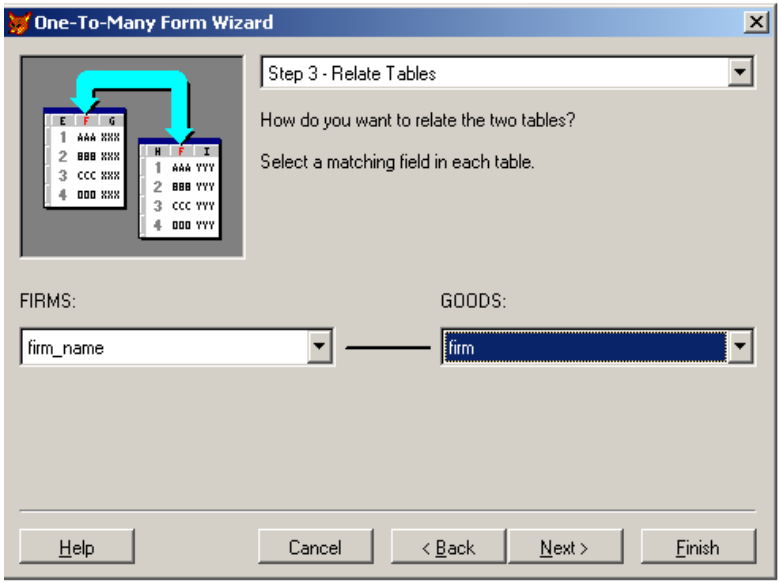


Рис. 4.9. Установка связи между таблицами

На четвертом шаге (см. рис. 4.4) выбираем понравившийся стиль отображения формы и вид кнопок, на пятом шаге – указываем поле или несколько полей, по которым будет проводиться сортировка данных в форме (окна аналогичны приведенным на рис. 4.4, рис. 4.5).

На шестом шаге задаем заголовок формы «*Товары в наличии*» в поле ***Type a title for your form***, а также указываем действие с созданное формой. Нажимаем кнопку ***Finish.***

В результате ввода информации в эти шесть окон мастер формы автоматически создаст форму для редактирования данных из двух связанных таблиц.

Пример формы «один-ко-многим» приведен на рис. 4.10.

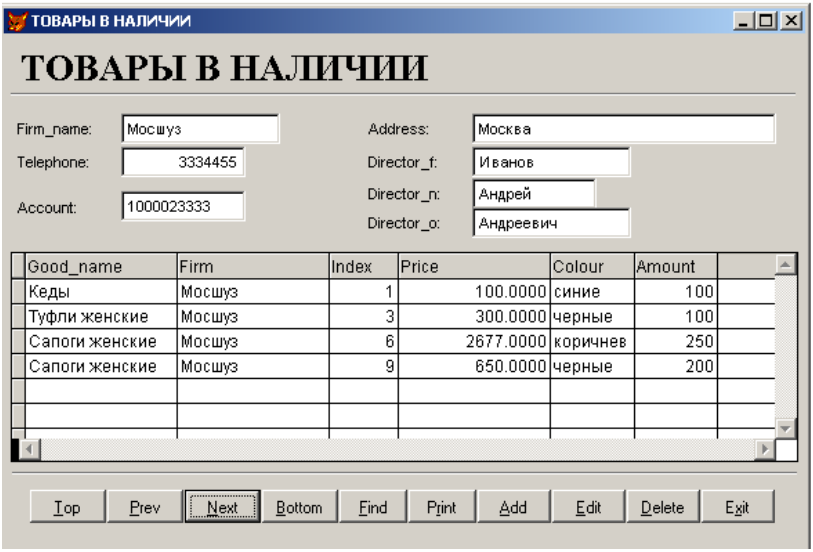


Рис. 4.10. Форма со связью «один-ко-многим», созданная с помощью мастера форм

4.1.2. Запуск формы

Запустить сохраненную форму можно несколькими способами:

- кнопка ***Run*** на стандартной панели инструментов;
- пункт меню ***Run Form*** из меню ***Form***;
- ввод в командном окне команды

```
DO FORM <имя_формы_без_расширения>.
```

Запустим форму по таблице goods, созданную в п.4.1.1.1, из командного окна командой `DO FORM goods`.

Результат выполнения команды приведен на рис. 4.11.

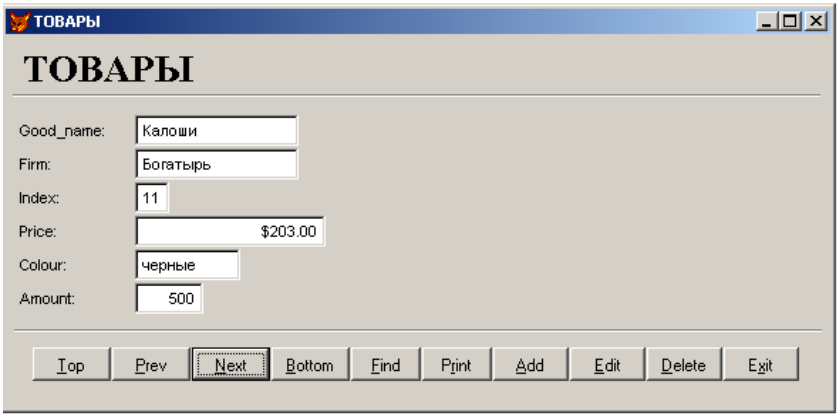


Рис. 4.11. Форма goods, созданная с помощью мастера форм

В нижней части окна формы расположен элемент управления из библиотеки классов Visual FoxPro – группа командных кнопок для управления источником данных. Эти кнопки позволяют перемещаться по таблице, выполнять поиск, удалять записи или вставлять новые записи.

При создании форм источник данных записывается в среде данных формы. Источником данных могут быть таблицы, представления или отношения. Среда данных сохраняется вместе с формой. Visual FoxPro автоматически открывает таблицы и представления, включенные в среду, при запуске формы и закрывает их при закрытии или освобождении файла формы.

По умолчанию (если иного не определено) все таблицы открываются для монопольного (эксклюзивного) использования, т.е. доступ к таблице разрешается только открывшему ее «пользователю», а остальные «пользователи» не имеют доступа к таблице.

Поэтому при одновременном открытии таблицы командой `USE` и запуске формы, в источнике данных которой записана эта таблица, может возникнуть ошибка (рис. 4.12 или рис. 4.13):

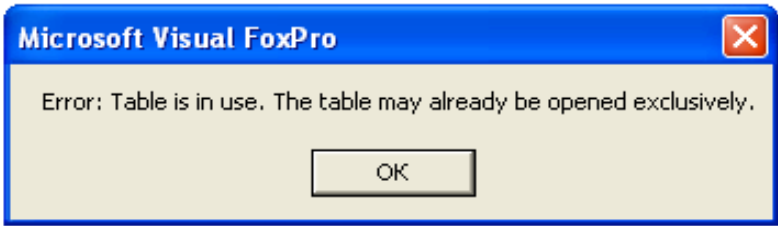


Рис. 4.12. Ошибка при совместном открытии таблицы и формы

Рис. 4.13. Ошибка при совместном открытии формы и таблицы

Для устранения этой ошибки перед запуском формы, использующей таблицу из некоторой рабочей области, необходимо закрыть таблицу в этой рабочей области (например, командой `CLOSE TABLES`). И наоборот – прежде, чем открыть таблицу командой `USE`, следует закрыть использующую ее форму.

4.1.3. Модификация формы

Для внесения изменений в созданную форму в командном окне вводим команду `MODIFY FORM <имя_формы>`. Visual FoxPro откроет окно конструктора формы и соответствующий ему набор инструментов (рис. 4.14).

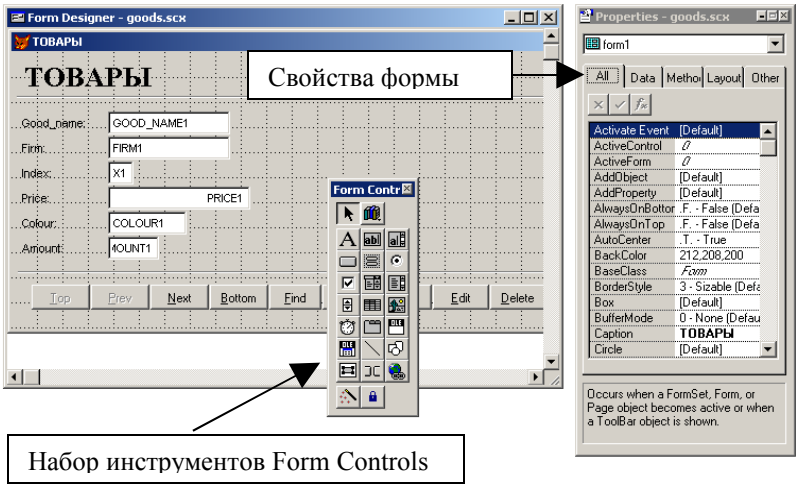


Рис. 4.14. Окно конструктора формы

Набор инструментария ***Form Controls***, содержит следующие кнопки (слева направо, сверху вниз) [3]:

Select Object – выбор объекта;
View Classes – загрузка других библиотек класса;
Label – элемент управления «метка»;
Text Box – элемент управления «текстовый блок»;
Edit Box – элемент управления «многострочное текстовое поле»;
Command Button – элемент управления «командная кнопка»;
Command Group – элемент управления «группа командных кнопок»;
Option Group – элемент управления «группа радиокнопок»;
Check Box – элемент управления «переключатель»;
Combo Box – элемент управления «раскрывающийся список»;
List Box – элемент управления «панель списка»;
Spinner – элемент управления «счетчик»;
Grid – элемент управления «окно таблицы»;
Image – элемент управления «рисунок»;
Timer – элемент управления «таймер»;
PageFrame – элемент управления «страничный блок»;
ActiveX Control (OLE Control) – элемент управления «контейнер OLE»;

ActiveX Bound Control (OLE Bound Control) – элемент управления «внедренный объект OLE»;

Line – элемент управления «линия»;

Shape – элемент управления «контур»;

Container – контейнер;

Separator – элемент управления «разделитель»;

Builder Lock – режим вызова построителя объектов;

Button Lock – переключение режима многократного/однократного размещения объектов.

Необходимым элементом, используемым при разработке экранных форм, является окно свойств (***Properties***). С помощью этого окна можно редактировать свойства, события и методы экранной формы и ее объектов.

Окно свойств можно условно разделить на две части: в левой, на сером фоне, перечислены названия всех свойств, методов и событий объектов, а в правой, на белом фоне, – значения этих свойств (событий, методов).

Окно свойств имеет пять вкладок [2]:

All – отображает все свойства, методы и события для данного объекта;

Data – отображает только те свойства, которые имеют отношение к данным;

Methods – отображает все методы и события объекта;

Layout – отображает свойства, которые «отвечают» за отображение объекта на экране;

Other – другие свойства объекта.

Изменим форму goods, добавим новую кнопку с функцией просмотра всех поставщиков. Выберем элемент управления ***Command Button*** и разместим его на форме (рис. 4.15). Для изменения названия кнопки в окне ***Properties*** изменим свойство ***Caption*** для кнопки (это свойство отвечает за подпись на кнопке).

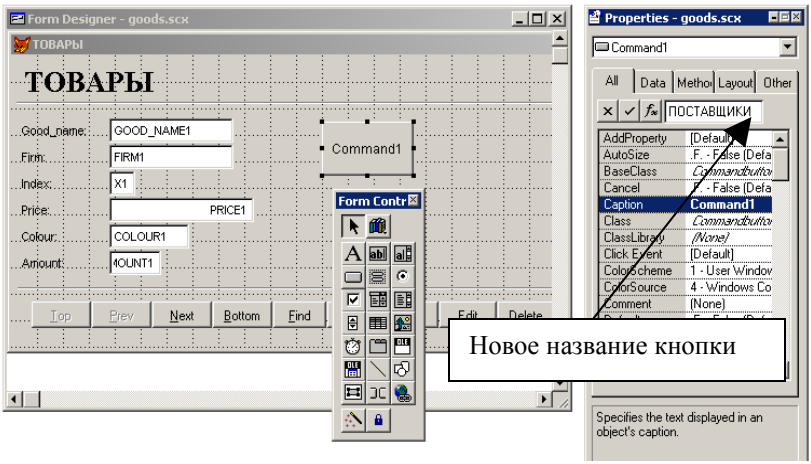


Рис. 4.15. Добавление нового элемента ***Command Button***

По правой кнопке мыши выбираем пункт ***Code*** (или на вкладке ***Methods*** выбираем событие – нажатие кнопки) и в открывшемся окне вводим текст программы, выполняющейся при нажатии на кнопку:

```
BROW FIELDS FIRM_NAME FONT «ARIAL»,15 .
```

Завершив модификацию формы, сохраним форму и закроем окно конструктора форм, запустим ее и убедимся в работоспособности добавленной кнопки.

4.1.4. Удаление формы

Удалить файл формы с диска можно с помощью команды

```
DELETE FILE <имя_формы>.SCX
```

4.2. Работа с отчетами

Отчеты используются для отображения информации, содержащейся в БД, в многостраничных выходных документах и позволяют осуществлять в нем необходимую группировку данных, отображать итоговые и расчетные данные.

4.2.1. Создание отчетов

Visual FoxPro предлагает три пути создания отчетов [3, 6]:

- создание простого или многотабличного отчета с применением мастера отчета (***Report Wizard***);
- создание простого отчета для одной таблицы с применением быстрого формирования отчета (***Quick Report***);
- изменение существующего отчета или создание нового отчета в окне проектирования отчета ***Report Designer***.

Рассмотрим создание отчета с помощью мастера создания отчетов.

Создание отчета через мастер создания отчетов аналогично созданию форм через мастер создания форм.

Мастер отчетов Visual FoxPro предлагает выбрать один из трех видов формируемого отчета [3, 6]:

- ***Group/Total Report Wizard*** (отчет с группируемыми данными и частичными суммами);
- ***One-to-Many Report Wizard*** (отчет с данными из двух связанных таблиц);
- ***Report Wizard*** (простой многоколонный отчет).

4.2.1.1. Создание отчета по одной таблице с помощью мастера

Создадим простой отчет по таблице goods.

Для этого выбираем пункт меню ***File→New***. В окне создания нового объекта (см. рис. 2.2) выбираем опцию создания отчета (***Report***) и нажимаем на кнопку мастера отчетов (***Wizard***). В появившемся окне ***Wizard Selection*** выбираем тип создаваемого отчета ***Report wizard*** (рис. 4.16).

Рис. 4.16. Выбор типа мастера отчетов

Так же, как и в мастере форм, мастер отчетов предлагает ответить на вопросы нескольких последовательных окон.

После нажатия кнопки ***OK*** в окне ***Wizard Selection***, появляется первое диалоговое окно мастера, в котором необходимо указать таблицу, для которой создается отчет и выбрать размещаемые в этом отчете поля. Аналогично созданию формы, выбираем таблицу goods, по которой и будем создавать отчет. В список ***Selected fields*** переносим поля для отображения в отчете из списка ***Available fields*** (рис. 4.17).

На втором шаге необходимо указать поля, по которым будет осуществляться группировка данных в отчете (рис. 4.18).

Затем выбираем стиль отображения отчета (рис. 4.19), ориентацию страницы – книжная или альбомная (рис. 4.20) и поля, по которым требуется упорядочение данных в отчете (рис. 4.21).

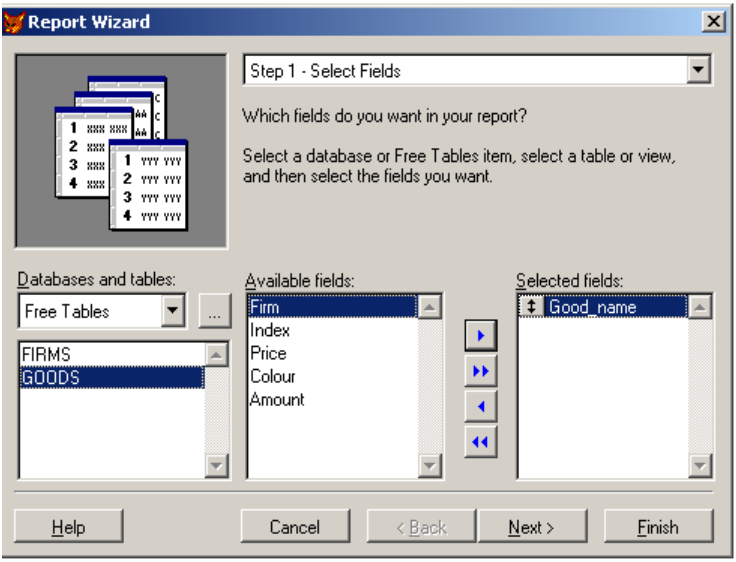


Рис. 4.17. Выбор полей таблицы для отображения в отчете

Рис. 4.18. Выбор полей для группировки данных в отчете

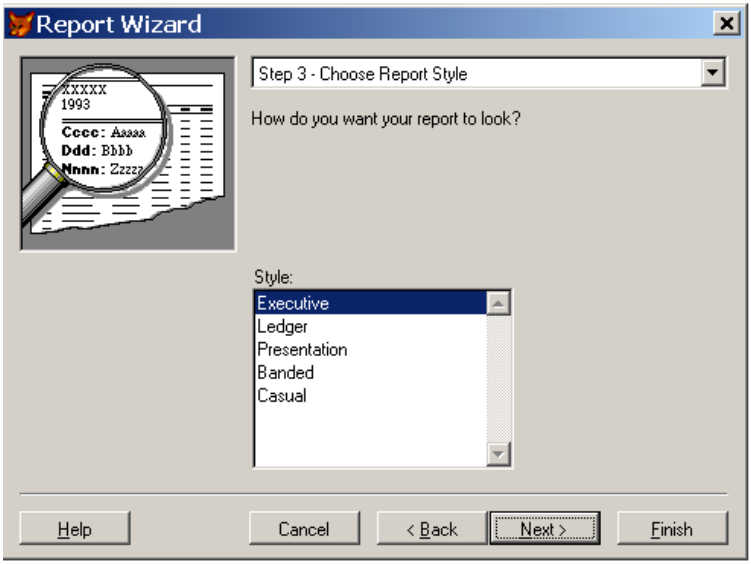


Рис. 4.19. Выбор стиля отчета

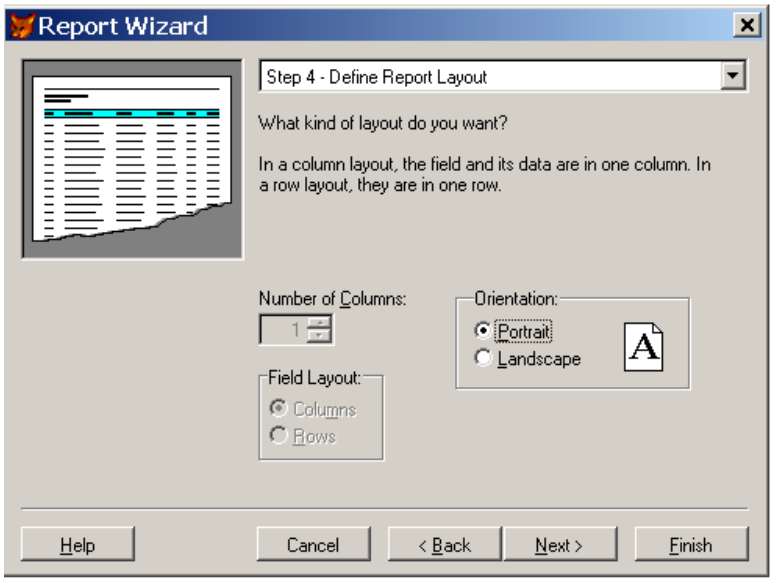


Рис. 4.20. Выбор ориентации страницы отчета

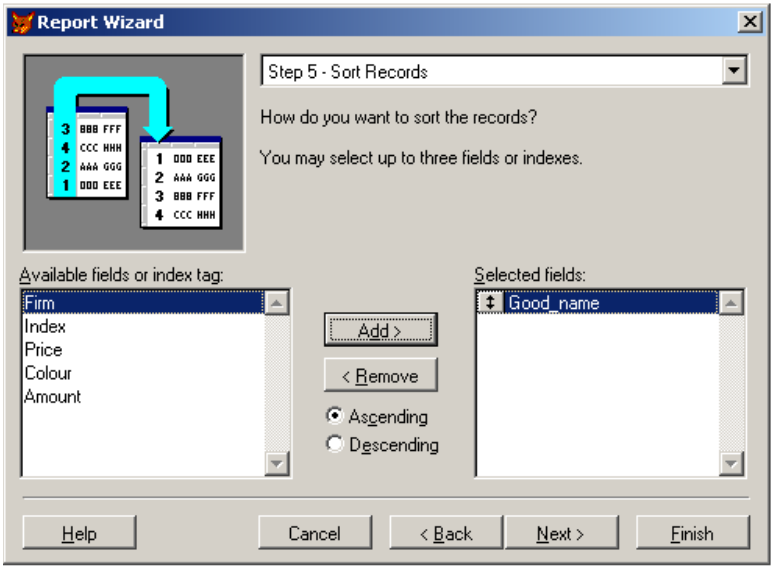


Рис. 4.21. Указание полей для сортировки данных в отчете

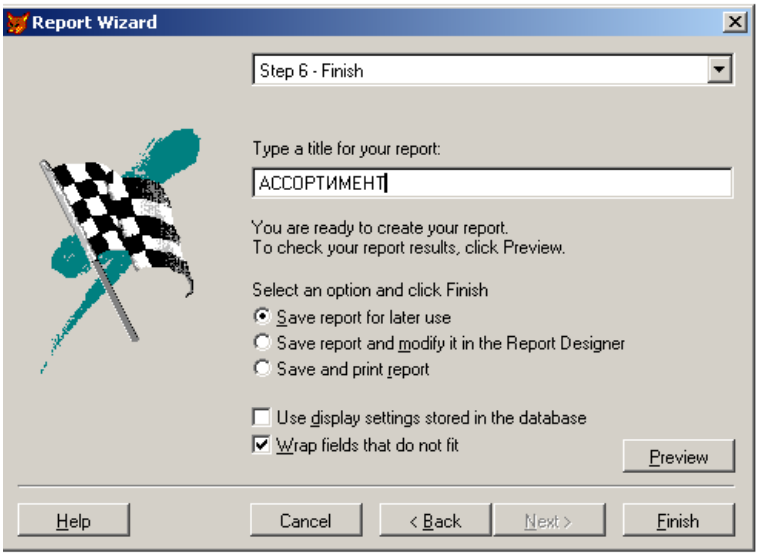


Рис. 4.22. Заключительный этап проектирования отчета

Задаем заголовок отчета в поле ***Type a title for your report***. На этом же шаге выбираем один из трех вариантов дальнейшей работы с отчетом (рис. 4.22):

- ***Save report and modify it in the Report Designer*** – созданный отчет сохраняется на диске под указанным именем (goods.frx) и открывается в конструкторе отчетов для модификации;
- ***Save report for later use*** – созданный отчет сохраняется на диске под указанным именем;
- ***Save and print report*** – созданный отчет сохраняется на диске под указанным именем и выводится на печать.

В завершение этих действий Visual FoxPro автоматически сформирует отчет для указанной таблицы. Сформированный отчет записывается в файл с расширением .FRX. В дальнейшем сохраненный отчет можно изменять в конструкторе отчета ***Report Designer.***

4.2.1.2. Создание отчета по двум таблицам с помощью мастера

Создадим отчет на основе двух таблиц. Для этого выбираем пункт меню ***File→New,*** в диалоговом окне ***New*** (см. рис. 2.2) выбираем опцию создания отчета (***Report***) и нажимаем на кнопку мастера форм (***Wizard***). В появившемся диалоговом окне ***Wizard Selection*** выбираем тип создаваемого отчета ***One-to-Many Report Wizard*** (см. рис. 4.16).

После нажатия кнопки ***OK***, появляется первое диалоговое окно мастера, в котором необходимо указать родительскую таблицу для создаваемого отчета и выбрать поля этой таблицы, размещаемые в отчете (аналогично рис. 4.7). На данном шаге мы выбираем таблицу firms, которая и будет родительской. В список ***Selected fields*** мы переносим поля родительской таблицы, которые хотим разместить на создаваемой форме.

На следующем шаге (аналогично рис. 4.8) выбираем дочернюю таблицу (goods) и поля, которые будут размещены на форме от второй таблицы.

На третьем шаге (аналогично рис. 4.9) выбираются поля, по которым осуществляется связь между выбранными таблицами.

Четвертый шаг (аналогично рис. 4.21) – указываем поле, по которому идет сортировка данных в отчете.

На пятом шаге (аналогично рис. 4.19, рис. 4.20) выбираем стиль и ориентацию бумаги создаваемого отчета. На последнем этапе (аналогично рис. 4.22) задаем заголовок отчета, используя для этого поле ввода ***Type a title for your report***. На этом же шаге выбираем один из трех вариантов дальнейшей работы с отчетом.

Пример отчета по двум таблицам представлен на рис. 4.23.

ОТЧЕТ ПО ДВУМ ТАБЛИЦАМ

10/07/07

Firm Name: Intershoes LTD

Telephone: 11,001,100
Director F: Рабинович
Account: 3333300111

Good Name	Index	Colour	Price
Сапоги мужские	7	черные	1999.0000
Туфли мужские	10	черные	1450.0000

Firm Name: Vermishelle

Telephone: 44,552,211
Director F: Легран
Account: 997654321

Good Name	Index	Colour	Price
Шнурки	8	зеленые	15.0000

Рис. 4.23. Пример отчета по двум связанным таблицам

4.2.2. Печать отчета

Для просмотра отчета используется пункт меню ***Preview*** из контекстного меню или одноименная кнопка на стандартной панели инструментов. Еще один способ – выполнить команду из командного окна `REPORT FORM <имя_отчета> PREVIEW`. При выборе данной команды отчет открывается для предварительного просмотра. Например, откроем окно предварительного просмотра отчета goods:

```
REPORT FORM GOODS PREVIEW
```

Результат выполнения команды приведен на рис. 4.24.

Рис. 4.24. Окно предварительного просмотра отчета

Вывод отчета на печать тоже осуществляется несколькими способами:

- пункт меню ***Print*** из системного меню ***File***;
- пункт меню ***Run Report*** из меню ***Report***;
- команда `REPORT FORM <имя_отчета> TO PRINTER` из командного окна.

4.2.3. Модификация отчета

Чтобы редактировать отчет, необходимо либо через пункт меню ***File→Open→Report*** открыть необходимый отчет, либо выполнить команду в командном окне: `MODIFY REPORT <имя_отчета_без_расширения>`. В результате будет открыт конструктор отчетов (***Report Designer***), в котором осуществляется редактирование отчета (рис. 4.25).

Вся рабочая область конструктора отчетов по умолчанию разделена на три полосы, ограничиваемые разделительными строками. Наименование полосы отображается на разделительной строке, находящейся непосредственно под этой полосой. Также в отче-

те могут использоваться дополнительные полосы. Назначения полос приведены в табл. 4.1 [2, 6].

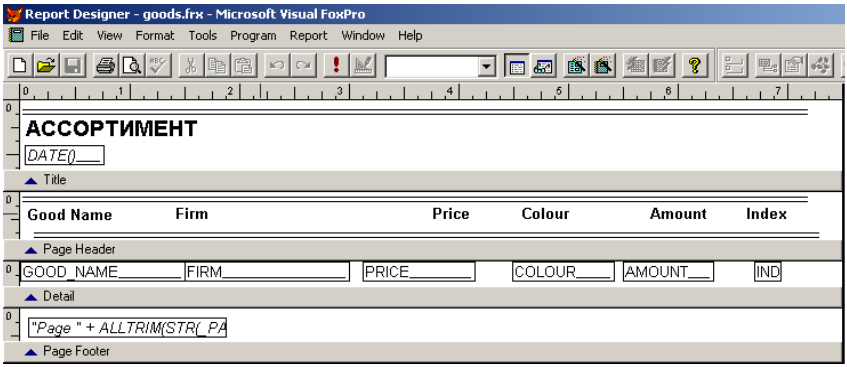


Рис. 4.25. Окно конструктора отчета

В областях отчета можно размещать поля данных, выбрав соответствующий элемент ***Field*** в наборе инструментария ***Report Controls*** (рис. 4.26) и указав мышью местоположение объекта в отчете.

При этом в зависимости от установленного режима ***Button Lock*** (многократное размещение объектов) выполняются следующие действия:

- по каждому щелчку мыши встраивается следующий объект;
- встраивается только один объект, который и остается выделенным.

Набор инструментария ***Report Controls*** (см. рис. 4.26), содержит следующие кнопки (слева направо, сверху вниз) [3, 5]:

Select Object – выбор объекта;
Label – метка;
Field – поле данных;
Line – линия;
Rectangle – прямоугольник;
Rounded Rectangle – прямоугольник со скругленными углами;
Picture/OLEBoundControl – рисунок;
Button Lock – переключение режима многократного размещения объектов.

Рис. 4.26. Инструментарий ***Report Controls***

Таблица 4.1. Назначение полос конструктора отчетов [2, 6]

Наименование полос	**Назначение**
Title	Здесь размещается информация, появляющаяся перед основным отчетом. Это может быть имя отчета или любые другие данные, помещаемые только вверху первой страницы отчета
Page Header	Данные, помещенные в полосу, печатаются в начале каждой страницы (верхний колонтитул). Например, название отчета, текущая дата или номер страницы
Group Header	При использовании группировки данных в отчете информация из этой области печатается перед печатью самой группы данных. Это заголовок для группы
Detail	Это полоса является основной и содержит данные полей из таблицы или результат вычислений над ними
Page Footer	Данные, помещенные в полосу, печатаются в конце каждой страницы (нижний колонтитул). Например, дата, номер страницы и итоговые значения по данным текущей страницы
Group Footer	Итоги по группе при использовании группировки данных
Summary	В итоговой части отчета содержится информация, повторяющаяся один раз после основного отчета и содержащая итоговые значения или заключительный текст

4.2.4. Удаление отчета

Удалить файл отчета с диска можно с помощью команды:

```
DELETE FILE <имя_отчета>.FRX
```

4.3. Работа с этикетками

4.3.1. Создание этикетки

Этикетки – это мини-отчеты, размещаемые по нескольку на одной странице (рис. 4.27). Этикетки, как и формы и отчеты, можно создавать с помощью мастера этикеток.

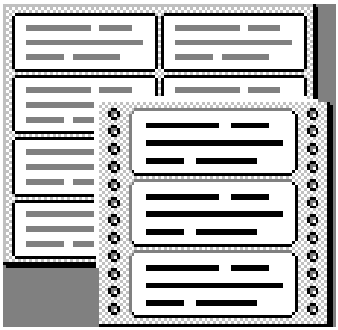


Рис. 4.27. Внешний вид этикетки

Создадим этикетки для таблицы товаров goods. Для этого выбираем пункт меню ***File→New***. В окне создания нового файла (см. рис. 2.2) выберем опцию создания этикеток ***Label*** и нажмем на кнопку мастера этикеток ***Wizard***.

В появившемся первом окне мастера необходимо выбрать источник данных, т.е. указать таблицу, из которой будут выбираться данные (goods) для создания этикетки (рис. 4.28).

На втором шаге из предложенного списка наиболее часто используемых размеров этикеток нужно выбрать подходящий. Столбец ***Avery*** содержит наименования этикеток. В столбце ***Dimensions*** (Размеры) отображаются размеры этикетки, а в столбце ***Columns*** (Колонки) – количество колонок этикетки, помещаемых на странице (рис. 4.29).

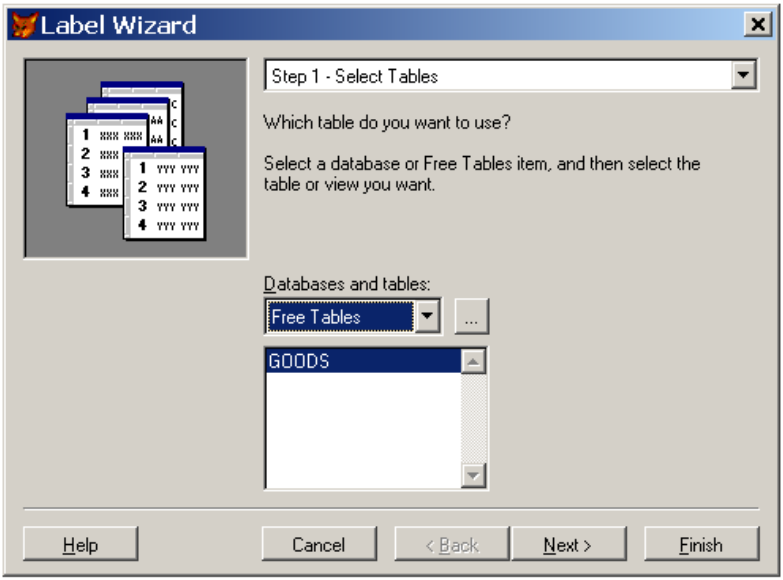


Рис. 4.28. Выбор таблицы при создании этикетки

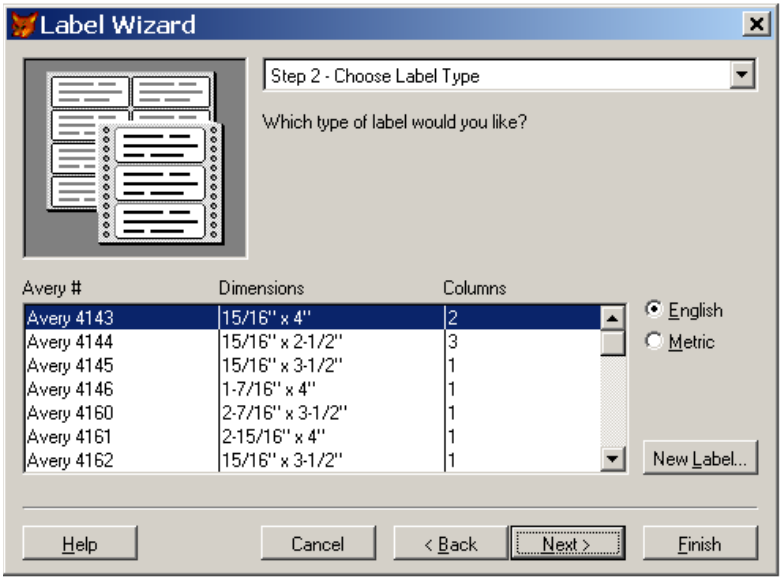


Рис. 4.29. Выбор размера этикетки

На третьем шаге создания этикетки необходимо расположить поля, знаки пунктуации (точка, запятая, дефис, двоеточие, пробел), сформировав текст этикетки (рис. 4.30).

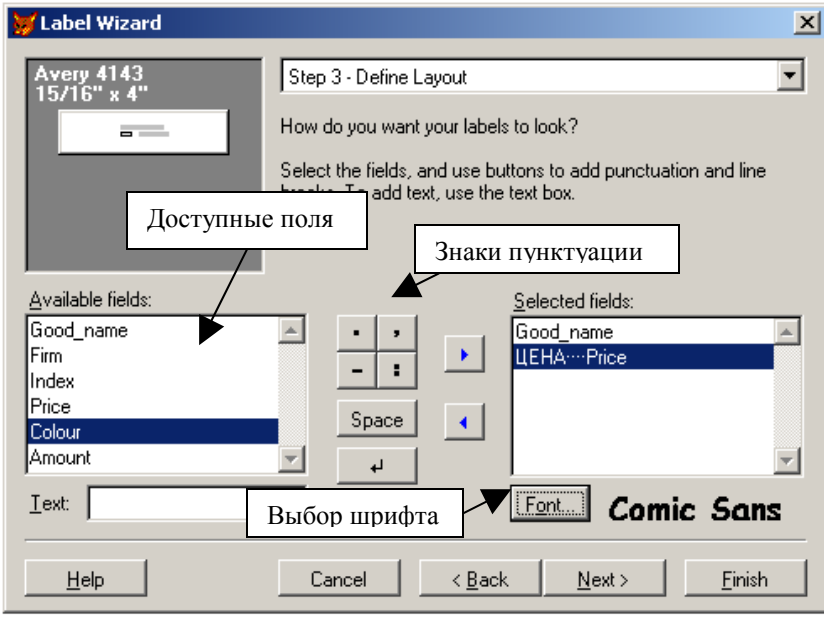


Рис. 4.30. Расположение полей на этикетке

На предпоследнем шаге следует задать порядок сортировки данных (аналогично рис. 4.21). На пятом, заключительном шаге (аналогично рис. 4.22) вариантов выбираем одну из предложенных опций:

- ***Save label and modify it in the Label Designer*** – созданная этикетка сохраняется на диске под указанным именем и открывается в конструкторе этикеток для модификации;
- ***Save label for later use*** – созданная этикетка сохраняется на диске под указанным именем;
- ***Save and print label*** – созданная этикетка сохраняется на диске под указанным именем и выводится на печать.

Сохраняем получившиеся этикетки под именем label_goods.lbx. Внешний вид этикетки представлен на рис. 4.31.

Рис. 4.31. Внешний вид этикетки

4.3.2. Печать этикетки

Этикетка аналогична отчету, поэтому команды предварительного просмотра и печати этикетки аналогичны командам предварительного просмотра и печати отчетов.

Для печати этикетки используются команды

```
LABEL FORM <имя_этикетки> TO PRINTER или
REPORT FORM <имя_этикетки>.lbx TO PRINTER.
```

Для предварительного просмотра используются команды

```
LABEL FORM <имя_этикетки> PREVIEW или
REPORT FORM <имя_этикетки>.lbx.
```

4.3.3. Модификация этикетки

Для изменения этикетки нужно открыть конструктор этикетки (***Label Designer***) либо через пункт меню (***File→Open→Label***), либо выполнить команду `MODIFY LABEL <имя_этикетки>` в командном окне. Редактирование этикетки аналогично редактированию отчета (см. п.4.2.3).

4.3.4. Удаление этикетки

Удалить файл этикетки с диска можно с помощью команды

```
DELETE FILE <имя_этикетки>.LBX.
```

4.4. Работа с меню

Visual FoxPro позволяет создавать два типа меню: обычное меню в виде строки, т.е. так называемую линейку главного меню (***Menu***), и всплывающее контекстное меню, т.е. самостоятельное ниспадающее меню (***Shortcut***). Описание меню хранится в файлах с расширениями MNX и MNT, сгенерированный текст программы меню хранится в файле с расширением MPR.

4.4.1. Создание меню

Создадим обычное меню в виде строки для нашего примера.

Для этого выбираем пункт меню ***File→New***, в диалоговом окне ***New*** (см. рис. 2.2) выбираем опцию создания меню (***Menu***) и нажимаем на кнопку ***New File***. В появившемся диалоговом окне ***New Menu*** выбираем тип создаваемого меню ***Menu*** (рис. 4.32)

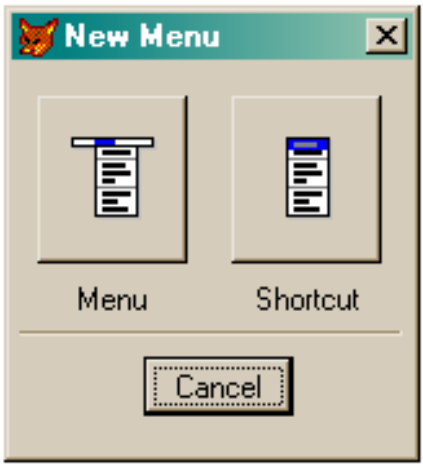


Рис. 4.32. Выбор типа меню

Окно ***New Menu*** также можно открыть через командное окно командой `CREATE MENU`.

В окне конструктора меню (рис. 4.33) необходимо определить пункты меню, определить подменю (если таковое предусматривается) и описать действия, выполняемые при выборе пунктов меню.

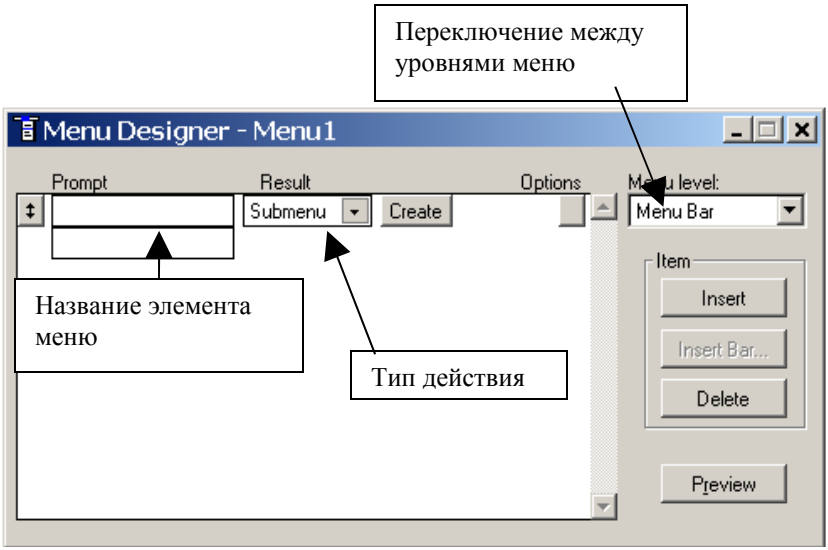


Рис. 4.33. Поля окна проектирования меню

В столбце ***Prompt*** записываются названия пунктов меню. Раскрывающийся список столбца ***Result*** позволяет определить тип действия при выборе пункта меню – команда (***Command***), команды системного меню (***Pad Name***), подменю (***Submenu***) или программа (***Procedure***). Раскрывающийся список ***Menu Level*** позволяет переключаться между уровнями меню. Кнопки, расположенные в правой части конструктора, имеют следующее назначение [2]:

Insert – добавляет новый пункт меню;

Insert Bar – позволяет разместить в пользовательском меню команды системного меню Visual FoxPro;

Delete – удаляет указанный пункт меню;

Preview – предварительный просмотр внешнего вида меню.

Пример заполненного окна проектирования меню и подменю приведены на рис. 4.34, рис. 4.35.

Примерное содержание меню приведено на рис. В1 и в табл. 4.2.

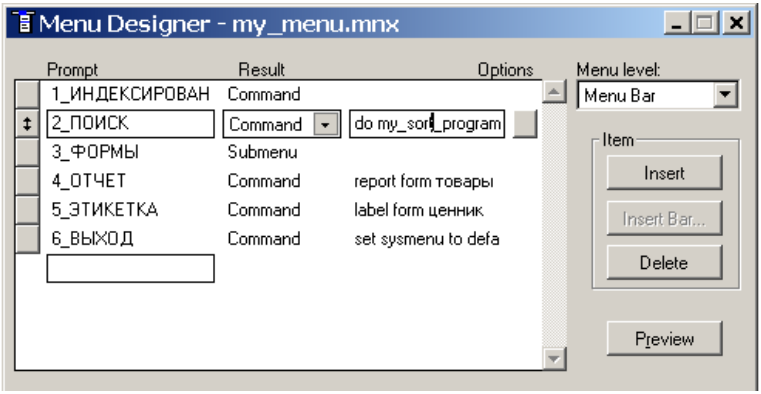


Рис. 4.34. Конструктор меню

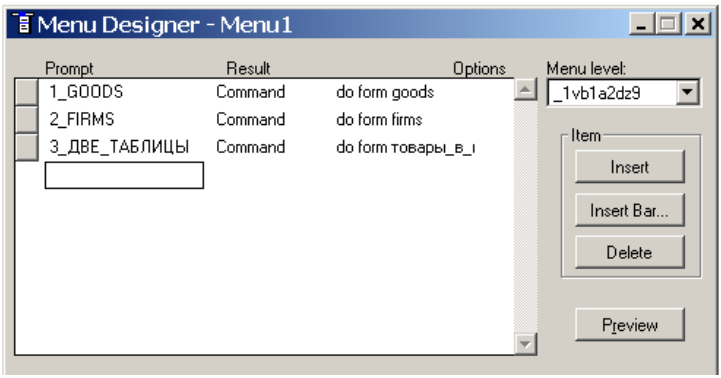


Рис. 4.35. Окно проектирования подменю

Таблица 4.2. Примерное содержание меню

Пункт меню	**Тип действия**	**Описание действия**
Поиск	Command	Запуск программы поиска
Индексирова-ние	Command	Запуск программы сортировки
Запуск форм	Submenu	Запуск форм
Отчет(ы)	Command (Submenu)	Запуск отчета (отчетов)
Этикетка(и)	Command (Submenu)	Запуск этикетки (этикеток)
Гистограмма	Command	Запуск программы построения гистограммы
Выход	Command	Выход в главное меню Visual Foxpro

4.4.2. Запуск меню

Для создания выполняемого файла (программы, запускающей меню) необходимо сгенерировать код программы, выбрав пункт меню ***Menu→Generate*** главного меню FoxPro (данный пункт меню доступен при активном окне проектирования меню). Программа, запускающая меню, имеет расширение .MPR. Запустить эту программу из командного окна можно командой

`DO <имя_меню>.MPR` (рис. 4.36).

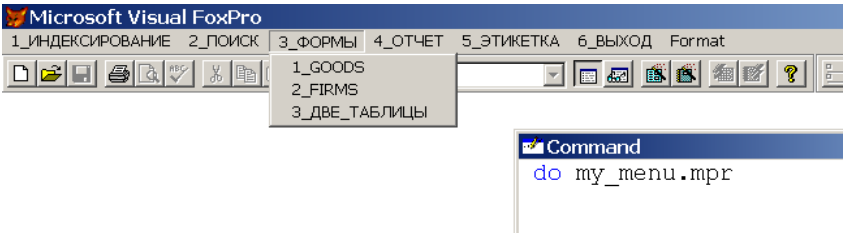


Рис. 4.36. Вид линейки меню

Для возврата к основному меню Visual FoxPro необходимо использовать команду: `SET SYSMENU TO DEFAULT`.

4.4.3. Клавиши быстрого доступа и заголовки пунктов меню

Для ускорения доступа к функциям меню используются клавиши быстрого доступа. Одновременное нажатие клавиши быстрого доступа и клавиши ***Alt*** активизирует соответствующий пункт меню. Клавиша доступа обозначается подчеркнутой буквой в заголовке пункта меню или подменю.

Если вы не назначили клавишу доступа для заголовка меню или элемента меню, то Visual FoxPro автоматически назначит в качестве клавиши доступа первую букву названия пункта меню.

В случае, если заголовки пунктов меню начинаются с букв русского языка, Visual FoxPro автоматически назначит в качестве клавиши доступа сочетание клавиши ***Alt*** и соответствующей буквы кириллицы, поэтому при запуске меню могут возникнуть ошибки.

Для примера создадим меню, состоящее из единственного пункта «Пункт1» (рис. 4.37).

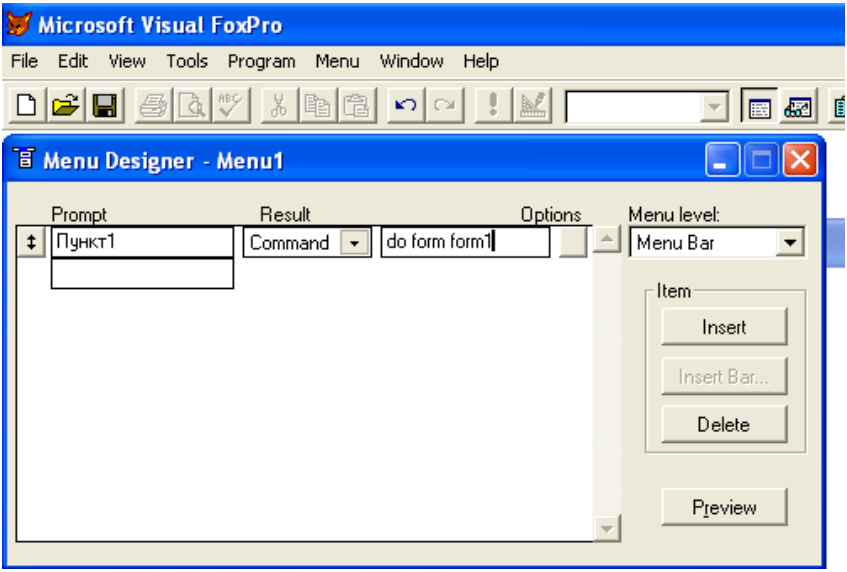


Рис. 4.37. Создание меню

Сгенеровав код программы меню (***Menu→Generate***), запустим меню командой `DO menu1.mpr`. Так как Visual FoxPro не сможет создать клавишу быстрого доступа ***«Alt»+П*** , при запуске меню возникнет ошибка (рис. 4.38).

В этом случае следует выбрать один из следующих вариантов:

- начинать именовать пункты меню с цифры;
- изменить клавиши быстрого доступа;
- закомментировать использование клавиш быстрого доступа в сгенерированном файле меню **`*.mpr`**.

Чтобы изменить клавишу быстрого доступа, необходимо нажать на кнопку ***Options*** напротив соответствующего пункта меню, откроется диалоговое окно ***Prompt Options*** (рис. 4.39).

Далее необходимо установить курсор в поле ***Key Label*** и нажать на клавиатуре сочетание клавиш быстрого доступа к данному пункту меню, в поле ***Key Text*** можно ввести краткое пояснение. Например, ***Cntr+G*** (рис. 4.40). После внесения изменений необходимо снова сгенерировать код программы меню (***Menu→Generate***). На рис. 4.41 представлен код сгенерированного меню с новой клавишей быстрого доступа.

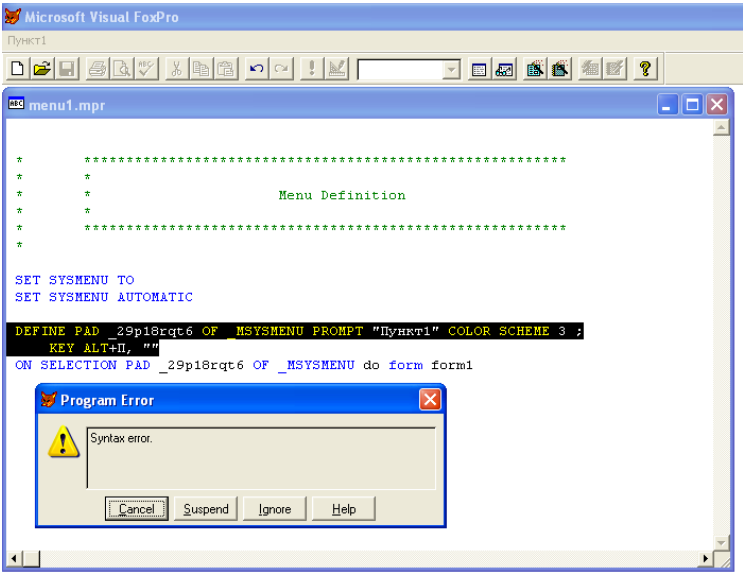


Рис. 4.38. Ошибка при запуске меню

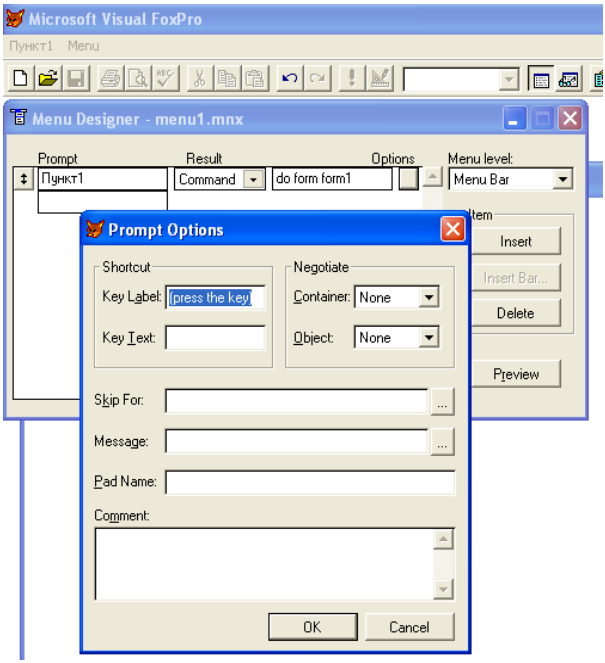


Рис. 4.39. Окно для назначения клавиши быстрого доступа на пункт меню

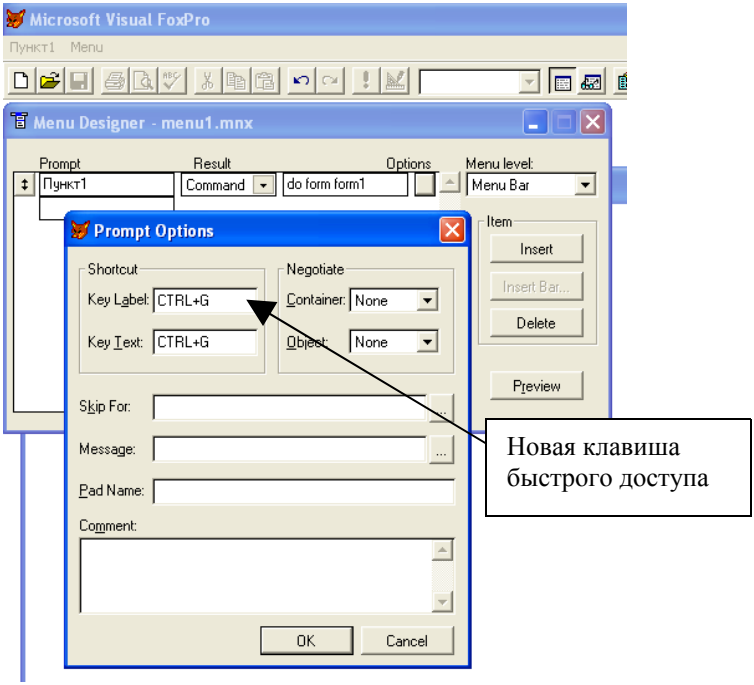


Рис. 4.40. Назначение клавиши быстрого доступа на пункт меню

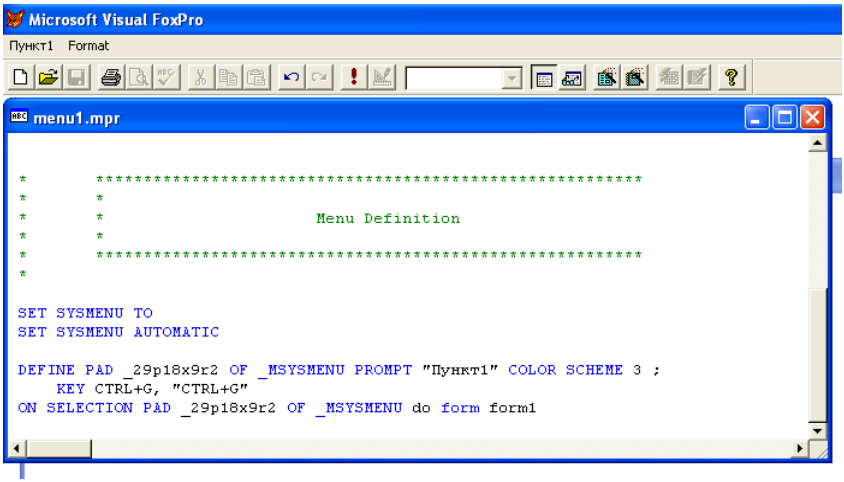


Рис. 4.41. Код сгенерированного меню с новой клавишей быстрого доступа

Чтобы закомментировать использование клавиш быстрого доступа в сгенерированном файле меню ***.mpr** , необходимо открыть код файла меню (командой `MODIFY COMMAND menu1.mpr`) и поставить знаки комментария перед соответствующей строкой (рис. 4.42). Однако этот способ плох тем, что комментарии необходимо будет проставлять каждый раз после генерации файла меню.

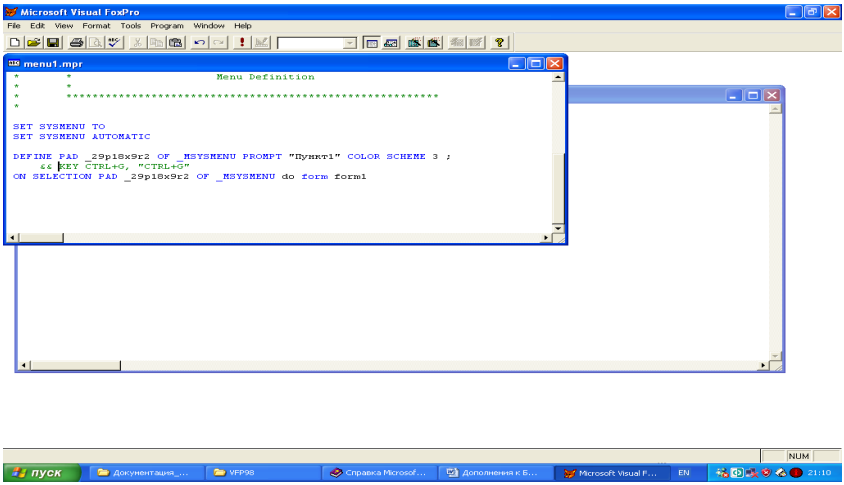


Рис. 4.42. Комментирование строк в сгенерированном файле меню

4.4.4. Модификация меню

Для внесения изменений в созданное меню в командном окне вводим команду `MODIFY MENU <имя_меню>`. Visual FoxPro откроет конструктор меню (см. рис. 4.34). После внесения изменений в меню необходимо повторно сгенерировать код программы меню.

4.4.5. Удаление меню

Удалить файл меню с диска можно с помощью команды `DELETE FILE <имя_меню>.MPR`

Заключение

В ходе лабораторной работы студенты должны научиться создавать формы, отчеты и этикетки с помощью соответствующих мастеров, модифицировать их и просматривать на экране.

Вопросы для самоконтроля

1. Для чего служит экранная форма (Form)?
2. Как изменить существующие формы, отчеты или этикетки?
3. Чем отличаются отчет и этикетка?
4. Какие виды меню поддерживаются Visual FoxPro?
5. Как изменить созданное меню?
6. В каком случае в качестве действия, выполняемого при нажатии пункта меню, в конструкторе меню создается процедура?
7. В какую полосу конструктора отчетов заносятся данные?
8. Какой командой можно распечатать отчет? Этикетку?

Практическое занятие 5
Структурированный язык запросов SQL

Цель работы: изучить SQL-команды и применять их на практике.

5.1. Общая характеристика языка

Язык SQL (Structured Query Language) предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также некоторых сопутствующих операций, и не содержит операторов управления, организации подпрограмм, ввода-вывода и т. п. Таким образом, SQL не обладает функциями полноценного языка разработки, т.е. является *непроцедурным* языком. В связи с этим SQL автономно не используется, обычно его включают в состав средств разработки программ СУБД (например, FoxPro СУБД Visual FoxPro, ObjectPAL СУБД Paradox, Visual Basic for Applications СУБД Access), причем в различных СУБД состав операторов SQL может несколько отличаться [1].

Язык SQL является первым и пока единственным стандартным языком работы с базами данных, который получил достаточно широкое распространение [7]. Практически все крупнейшие разработчики СУБД в настоящее время создают свои продукты с использованием языка SQL либо интерфейса SQL, и большинство таких компаний участвуют в работе, по меньшей мере, одной организации, которая занимается разработкой стандартов этого языка. Язык SQL также принят в качестве Федерального стандарта обработки информации (Federal Information Processing Standard – FIPS), который должен соблюдаться в СУБД для получения разрешения продавать ее на территории США. Хотя исходные концепции языка SQL были разработаны корпорацией IBM, его важность очень скоро подтолкнула и других разработчиков к созданию собственных реализаций. В настоящее время на рынке доступны буквально сотни продуктов, построенных на использовании языка SQL, причем постоянно приходится слышать о выпуске все новых и новых версий [7].

В современных СУБД с интерактивным интерфейсом можно создавать запросы, используя другие средства, например QBE (Query By Example – запрос по образцу)[2]. Запрос по образцу – это интерактивное средство для выбора данных из одной или нескольких таблиц. При формировании запроса Вам необходимо указать критерии выборки записей в исходной таблице. При этом вместо того, чтобы печатать предложения на специальном языке, необходимо просто заполнить бланк запроса, который располагается в окне конструктора запросов. Метод формирования запроса путем заполнения бланка прост для изучения и понимания. Он способствует эффективному использованию возможностей FoxPro пользователями, имеющими даже минимальный навык работы с приложением или не имеющими его вовсе.

Однако по сравнению с QBE применение SQL зачастую позволяет повысить эффективность обработки данных в базе. Например, при подготовке запроса в среде FoxPro можно перейти из окна конструктора запросов (формулировки запроса по образцу на языке QBE) в окно с эквивалентным оператором SQL. Подготовку нового запроса путем редактирования уже имеющегося в ряде случае проще выполнить путем изменения оператора SQL.

Основным назначением языка SQL (как и других языков для работы с базами данных) является подготовка и выполнение запросов. В результате выборки данных из одной или нескольких таблиц может быть получено множество записей, называемое представлением.

Представление по существу является таблицей, формируемой в результате выполнения запроса. Можно сказать, что оно является разновидностью хранимого запроса. По одним и тем же таблицам можно построить несколько представлений. Само представление описывается путем указания идентификатора представления и запроса, который должен быть выполнен для его получения [1].

5.2. Основные операторы языка

Опишем минимальное подмножество языка SQL, опираясь на его реализацию в стандартном интерфейсе ODBC (Open Database

[2] Подробнее о запросах по образцу см. [13-14].

Connectivity – совместимость открытых баз данных) фирмы Microsoft.

Операторы языка SQL можно условно разделить на два подъязыка: язык определения данных (Data Definition Language – DDL) и язык манипулирования данными (Data Manipulation Language – DML) [1, 8].

Основные операторы языка SQL, поддерживаемые Visual FoxPro, представлены в табл. 5.1.

Таблица 5.1. Операторы языка SQL

Вид	Название	Назначение
DDL	CREATE TABLE	Создание таблицы
	CREATE CURSOR	Создание временной таблицы
	ALTER TABLE	Изменение структуры таблицы
	CREATE SQL VIEW	Создание представления
DML	SELECT	Выборка записей
	UPDATE	Изменение записей
	INSERT	Вставка новых записей
	DELETE	Удаление записей

SQL оператор состоит из зарезервированных слов, а также из слов, определяемых пользователем. Зарезервированные слова являются постоянной частью языка SQL и имеют фиксированное значение. Их следует записывать в точности так, как это установлено, и нельзя разбивать на части для переноса из одной строки в другую. Слова, определяемые пользователем, задаются самим пользователем и представляют собой имена различных объектов базы данных. Слова в операторе размещаются в соответствии с установленными синтаксическими правилами [8].

Для определения формата SQL-операторов мы будем применять следующую расширенную форму BNF-нотации (Backus Naur Form).

- Прописные буквы будут использоваться для записи зарезервированных слов и должны указываться в операторах точно так же, как это будет показано.
- Строчные буквы будут использоваться для записи слов, определяемых пользователем.

- Вертикальная черта (|) указывает на необходимость выбора одного из нескольких приведенных значений – например, a | b | c.
- Фигурные скобки определяют обязательный элемент – например, {a}.
- Квадратные скобки определяют необязательный элемент – например, [a].
- Многоточие используется для указания необязательных возможности повторения конструкции, от нуля до нескольких раз – например, {a|b} [,c...]. Эта запись означает, что после a или b может следовать от нуля до нескольких повторений c, разделенных запятыми.

Рассмотрим формат и основные возможности команд SQL, поддерживаемых СУБД Visual FoxPro. Полный синтаксис команд представлен в приложении 4. Несущественные операнды и элементы синтаксиса (например, принятое во многих системах программирования правило ставить «;» в конце оператора) будем опускать [1].

5.2.1. Оператор создания таблицы

Оператор создания таблицы имеет следующий формат:

```
CREATE TABLE <имя_таблицы>
  (<имя_поля> <тип_данных> [NOT NULL]
  [,<имя_поля> <тип_данных> [NOT NULL]]...).
```

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного поля с указанием типа данных, хранимых в этом поле.

При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция `NOT NULL` (не пустое) служит именно таким целям и для поля таблицы означает, что в этом поле должно быть определено значение [1].

Пример 1. Создание таблицы

Пусть требуется создать таблицу goods описания товаров, имеющую поля: good − наименование товара, firm_name − название фирмы-производителя, index − уникальный индекс на складе, price − цена. Оператор определения таблицы будет иметь следующий вид:

```
CREATE TABLE goods
  (good_name CHAR(15) NOT NULL,
  firm CHAR(15) NOT NULL,
  index NUMERIC(2) NOT NULL,
  price CURRENCY)
```

5.2.2. Оператор изменения структуры таблицы

Оператор изменения структуры таблицы имеет формат вида:

```
ALTER TABLE <имя_таблицы>
  {ADD, MODIFY, DROP} <имя_поля> [<тип_данных>]
  [NOT NULL]
  [{ADD,DROP} PRIMARY KEY <выражение>
      TAG <имя_индекса> ]
  [{ADD, DROP} UNIQUE <выражение>
     [TAG <имя_индекса> ]]
   [{ADD, DROP} FOREIGN KEY [<выражение> ]
     TAG <имя_индекса> ]
   [RENAME COLUMN <старое_имя_поля>
    TO <новое_имя_поля>]
```

Изменение структуры таблицы может состоять в добавлении (ADD), изменении (MODIFY) или удалении (DROP) одного или нескольких полей таблицы, создании или удалении первичных или уникальных ключей или теги внешних ключей, переименовании существующих полей. Правила записи оператора `ALTER TABLE` такие же, как и оператора `CREATE TABLE`. При удалении поля указывать `<тип_данных>` не нужно.

Пример 2. Добавление поля таблицы

Пусть в созданной ранее таблице goods необходимо добавить поле amount для хранения количества товара. Для этого следует записать оператор вида:

```
ALTER TABLE goods ADD amount INT.
```

Пример 3. Удаление поля

Удалим из таблицы goods поле amount :

```
ALTER TABLE goods DROP amount
```

Пример 4. Создание уникального индекса

Для таблицы firms создадим уникальный индекс account (номер банковского счета):

```
ALTER TABLE firms ADD UNIQ account TAG ind_firms
```

Пример 5. Удаление индекса

Удалим созданный в примере 4 индекс:

```
ALTER TABLE firms DROP UNIQ TAG ind_firms
```

Пример 6. Переименование поля

Переименуем поле address таблицы firms:

```
ALTER TABLE firms RENAME COLUMN address
                                TO registration
```

5.2.3. Оператор создания представления

Оператор создания представления имеет формат вида:

```
CREATE SQL VIEW [<имя_представления> ] [REMOTE]
  [CONNECTION <имя_связи> [SHARE]
  | CONNECTION <имя_источника_данных>]
  [AS <оператор_SELECT>]
```

Данный оператор позволяет создать представление. Если имена полей в представлении не указываются, то будут использоваться имена полей из запроса, описываемого соответствующим оператором `SELECT` [1].

Пример 7. Создание представления

Пусть база данных shop объединяет таблицы goods и firms. В таблице goods имеются поля: good (наименование товара), firm_name (название фирмы-производителя), index (уникальный индекс на складе), price (цена товара). В таблице firms имеются поля: firm_name (название фирмы), address (адрес), telephone (телефон), director_fam (фамилия директора), director_name (имя директора), director_otch (отчество директора), account (номер банковского счета). Таблицы связаны между собой по полю firm_name. Требуется создать представление repr с краткой информацией о товарах и их производителях: название товара, название фирмы-производителя, телефон фирмы. Оператор определения представления имеет вид:

```
OPEN DATABASE shop
CREATE SQL VIEW repr
  AS SELECT goods.good_name, goods.firm, firms.telephone
     FROM goods
     WHERE goods.firm = fims.firm_name
```

5.2.4. Оператор выборки записей

Оператор выборки записей имеет формат вида:

```
SELECT [ALL | DISTINCT] <список_данных>
  FROM <список_таблиц>
  [WHERE <условие_выборки>]
  [GROUP BY <имя_поля> [,<имя_поля>]... ]
  [HAVING <условие_поиска>]
  [ORDER BY <спецификация>[,<спецификация>]...]
```

Это наиболее важный оператор из всех операторов SQL. Функциональные возможности его огромны. Рассмотрим основные из них [1].

Оператор `SELECT` позволяет производить выборку и вычисления над данными из одной или нескольких таблиц. Результатом выполнения оператора является ответная таблица, которая может иметь (`ALL`), или не иметь (`DISTINCT`) повторяющиеся строки. По умолчанию в ответную таблицу включаются все строки, в том числе и повторяющиеся. В отборе данных участвуют записи одной или нескольких таблиц, перечисленных в списке операнда `FROM`.

Список данных может содержать имена полей, участвующих в запросе, а также выражения над полями. В простейшем случае в выражениях можно записывать имена полей, знаки арифметических операций (+, – ,*,/), константы и круглые скобки. Если в списке данных записано выражение, то наряду с выборкой данных выполняются вычисления, результаты которого попадают в новый (создаваемый) столбец ответной таблицы.

При использовании в списках данных имен полей нескольких таблиц для указания принадлежности поля некоторой таблице применяют конструкцию вида: `<имя_таблицы>.<имя_поля>`.

Операнд `WHERE` задает условия, которым должны удовлетворять записи в результирующей таблице. Выражение `<условие выборки>` является логическим. Его элементами могут быть имена полей, операции сравнения, арифметические операции, логические связки (И, ИЛИ, НЕТ), скобки, специальные функции LIKE, NULL, IN и т. д.

Операнд `GROUP BY` позволяет выделять в результирующем множестве записей группы. Группой являются записи с совпадающими значениями в полях, перечисленных за ключевыми словами

GROUP BY. Выделение групп требуется для использования в логических выражениях операндов WHERE и HAVING, а также для выполнения операций (вычислений) над группами.

В логических и арифметических выражениях можно использовать следующие групповые операции (функции): AVG (среднее значение в группе), MAX (максимальное значение в группе), MIN (минимальное значение в группе), SUM (сумма значений в группе), COUNT (число значений в группе).

Операнд HAVING действует совместно с операндом GROUP BY и используется для дополнительной селекции записей во время определения групп. Правила записи <условия_поиска> аналогичны правилам формирования <условия_выборки> операнда WHERE.

Операнд ORDER BY задает порядок сортировки результирующего множества. Обычно каждая <спецификация> представляет собой пару вида: <имя_поля> [ASC | DESC].

Замечание. Оператор SELECT может иметь и другие более сложные синтаксические конструкции. Одной из таких конструкций, например, являются так называемые подзапросы. Они позволяют формулировать вложенные запросы, когда результаты одного оператора SELECT используются в логическом выражении условия выборки операнда WHERE другого оператора SELECT.

Пример 8. Выбор записей

Выберем поля таблицы firms, содержащие информацию о названии фирмы (поле firm_name) и номере банковского счета (поле account):

```
SELECT firm_name, account FROM firms
```

Пример 9. Выбор с условием

Выберем названия фирм, расположенных в Москве (адрес фирмы содержится в поле address таблицы firms). Оператор SELECT для этого запроса можно записать так

```
SELECT firm_name FROM firms WHERE address=«Москва»
```

Пример 10. Выбор с группированием

Пусть требуется найти минимальное и максимальное количество товара (поле amount) на складе для каждого производителя (поле firm). Оператор SELECT для этого запроса имеет вид:

```
SELECT firm, MIN(amount), MAX(amount)
   FROM goods GROUP BY firm
```

5.2.5. Оператор изменения записей

Оператор изменения записей имеет формат вида:

```
UPDATE <имя_таблицы>
  SET <имя_поля> = (<выражение> , NULL }
  [, SET <имя_поля> = (<выражение> , NULL }... ]
  [WHERE <условие>]
```

Выполнение оператора `UPDATE` состоит в изменении значений в определенных операндом `SET` полях таблицы для тех записей, которые удовлетворяют условию, заданному операндом `WHERE`.

Новые значения полей в записях могут быть пустыми (`NULL`), либо вычисляться в соответствии с арифметическим выражением. Правила записи арифметических и логических выражений аналогичны соответствующим правилам оператора `SELECT`.

Пример 11. Изменение записей

Пусть необходимо изменить номер банковского счета (поле account таблицы firms) для фирмы «Скороход». Запрос, сформулированный с помощью оператора `SELECT`, может выглядеть так:

```
UPDATE firms SET account = «1000020022»
    WHERE firm_name = «Скороход».
```

5.2.6. Оператор вставки новых записей

Оператор вставки новых записей имеет форматы двух видов:

```
INSERT INTO <имя_таблицы>
  [(<список_полей>)]
  VALUES (<список_значений>)
```

и

```
INSERT INTO <имя_таблицы>
  [(<список_полей>)]
  <предложение_SELECT>.
```

В первом случае оператор `INSERT` предназначен для ввода новых записей с заданными значениями в полях. Порядок перечисления имен полей должен соответствовать порядку значений, пере-

численных в списке операнда VALUES. Если <список_полей> опущен, то в <списке_значений> должны быть перечислены все значения в порядке полей структуры таблицы.

Во втором формате оператор INSERT предназначен для ввода в заданную таблицу новых строк, отобранных из другой таблицы с помощью предложения SELECT.

Пример 12. Ввод записей

Добавим в таблицу goods строку, содержащую информацию о новом товаре. Для этого можно записать оператор вида:

```
INSERT INTO goods
 VALUES («Стельки», «Скороход», 12, 20.5).
```

5.2.7. Оператор удаления записей

Оператор удаления записей имеет формат вида:

```
DELETE FROM <имя_таблицы> [WHERE <условие>].
```

Результатом выполнения оператора DELETE является удаление (пометка к удалению) из указанной таблицы строк, которые удовлетворяют условию, определенному операндом WHERE. Если необязательный операнд WHERE опущен, т.е. условие отбора удаляемых записей отсутствует, удалению подлежат все записи таблицы.

Пример 13. Удаление записей

В связи с прекращением сотрудничества с фирмой «Vermishelle» требуется удалить запись о ней из таблицы firms. Оператор DELETE для этой задачи будет выглядеть так:

```
DELETE FROM firms
       WHERE firm_name= «Vermishelle».
```

5.2.8. Оператор создания временной таблицы

Оператор создания временной таблицы имеет следующий формат:

```
CREATE CURSOR <имя_таблицы>
  (<имя_поля> <тип_данных> [NOT NULL]
  [,<имя_поля> <тип_данных> [NOT NULL]]...).
```

Временная таблица существует только до ее закрытия. Временной таблицей можно манипулировать так же, как и любой другой таблицей.

Пример 14. Создание временной таблицы

Пусть требуется создать временную таблицу country – названия стран, в которых работают фирмы-производители товаров, имеющую поле: country_name − наименование страны. Оператор определения временной таблицы будет иметь следующий вид:

```
CREATE CURSOR country
 (country_name CHAR(25) NOT NULL)
```

Заключение

В результате выполнения лабораторной работы студенты изучат команды языка SQL, доступные в СУБД Visual FoxPro, и научатся использовать эти команды на практике.

Вопросы для самоконтроля

1. Почему SQL называют непроцедурным языком?
2. Что такое представление? Какой оператор SQL создает представление?
3. Какие параметры таблицы можно изменить с помощью оператора `ALTER TABLE`?
4. Какой оператор SQL добавляет строку в таблицу?
5. Для чего используется оператор `SELECT`?
6. Что является результатом выполнения оператора `SELECT`?
7. Что обозначает символ «*» в операторе `SELECT * FROM goods`?
8. Каким образом указывается принадлежность столбца некоторой таблице при действиях с несколькими таблицами?
9. Как изменить значение поля данных?
10. Какой оператор позволяет удалить записи из таблицы?

Список источников информации

1. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений / Под ред. проф. А.Д. Хомоненко. СПб.: КОРОНА принт, 2000.

2. Шапорев Д.С. Visual FoxPro. Уроки программирования. СПб.: БХВ-Петербург, 2005.

3. Баженова И.Ю. Visual FoxPro 6.0. М.: Диалог-МИФИ, 1999.

4. Максимов В. FoxPro: Советы начинающим. Часть I. http://www.foxclub.ru/articles/index.php?id=32.

5. Палюшкевич Ю.В. Система управления базами данных Microsoft Access. Шаг 15. Связи между таблицами базы данных. http://it.kgsu.ru/MSAccess/access15.html.

6. Пинтер Лес, Пинтер Джон. Visual FoxPro:уроки программирования./Пер. с англ. М.:Журнал «The Pinter FoxPro Letter» ТОО «Эдэль» совместно с издательским отделом «Русская редакция» ТОО «Channel Trading Ltd.», 1996.

7. Введение в язык SQL. Информационный портал CSIT. http:// www.csit.ws/doc/sqlbook/101.php.

8. Конноли Томас, Бегг Каролин, Страчан Анна. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд: Пер. с англ. М.: Издательский дом «Вильямс», 2001.

9. Справочник по языку. Microsoft Visual FoxPro. Microsoft Corporation, 1995.

10. Руководство пользователя. Microsoft Visual FoxPro. Microsoft Corporation, 1995.

11. Руководство разработчика. Microsoft Visual FoxPro. Microsoft Corporation, 1995.

12. Грабер М. Понимание SQL.
http://www.sql.ru/docs/sql/u_sql/index.shtml.

13. M.M. Zloof, «Query by Example», AFIPS Conference Proceedings, National Computer Conference 44, 1975, pp. 431-438 .

14. M.M. Zloof, «Query by Example. The Invocation and Definition of Tables and Forms», Proceedings of The International Conference on Very Large Data Bases, Boston, Massachusetts, September 22-24, 1975, pp. 1-24.

ПРИЛОЖЕНИЯ

Приложение 1
Определения нормальных форм

Таблица находится в *первой нормальной форме* (*1НФ*) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.

Таблица находится во *второй нормальной форме* (*2НФ*), если она удовлетворяет определению 1НФ и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом.

Таблица находится в *третьей нормальной форме* (*3НФ*), если она удовлетворяет определению 2НФ и ни одно из ее неключевых полей не зависит функционально от любого другого неключевого поля.

Таблица находится в *нормальной форме Бойса-Кодда* (*НФБК*), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от *возможного* ключа.

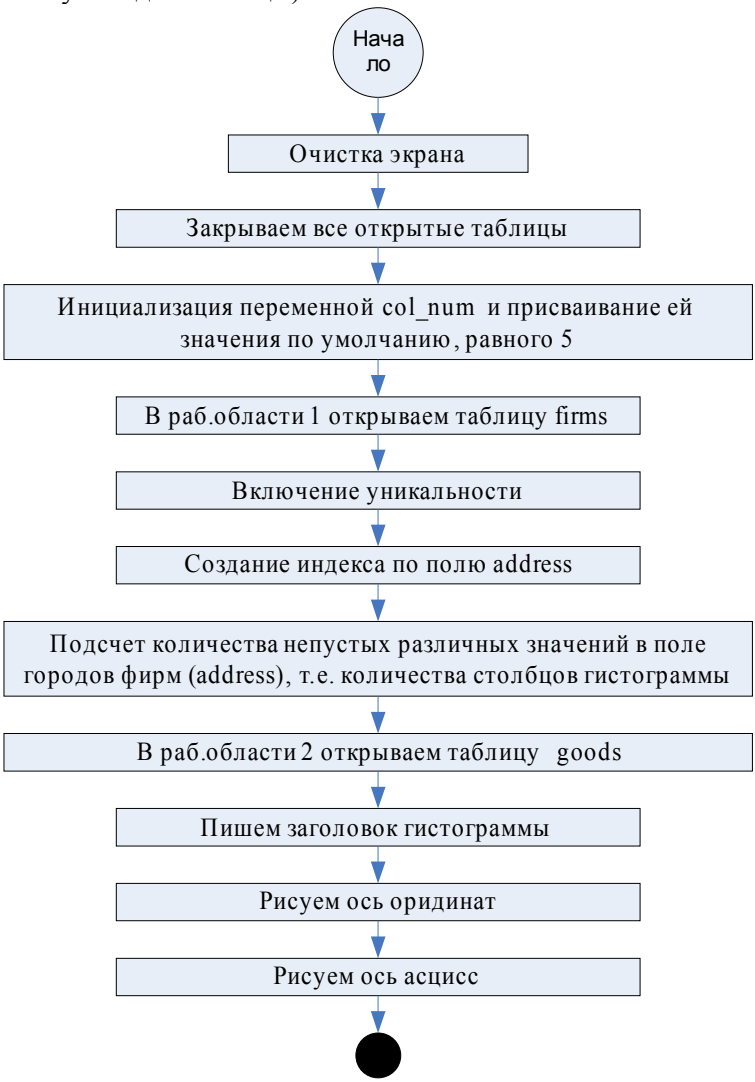
Таблица находится в *пятой нормальной форме* (*5НФ*) тогда и только тогда, когда в каждой ее полной декомпозиции[3] все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

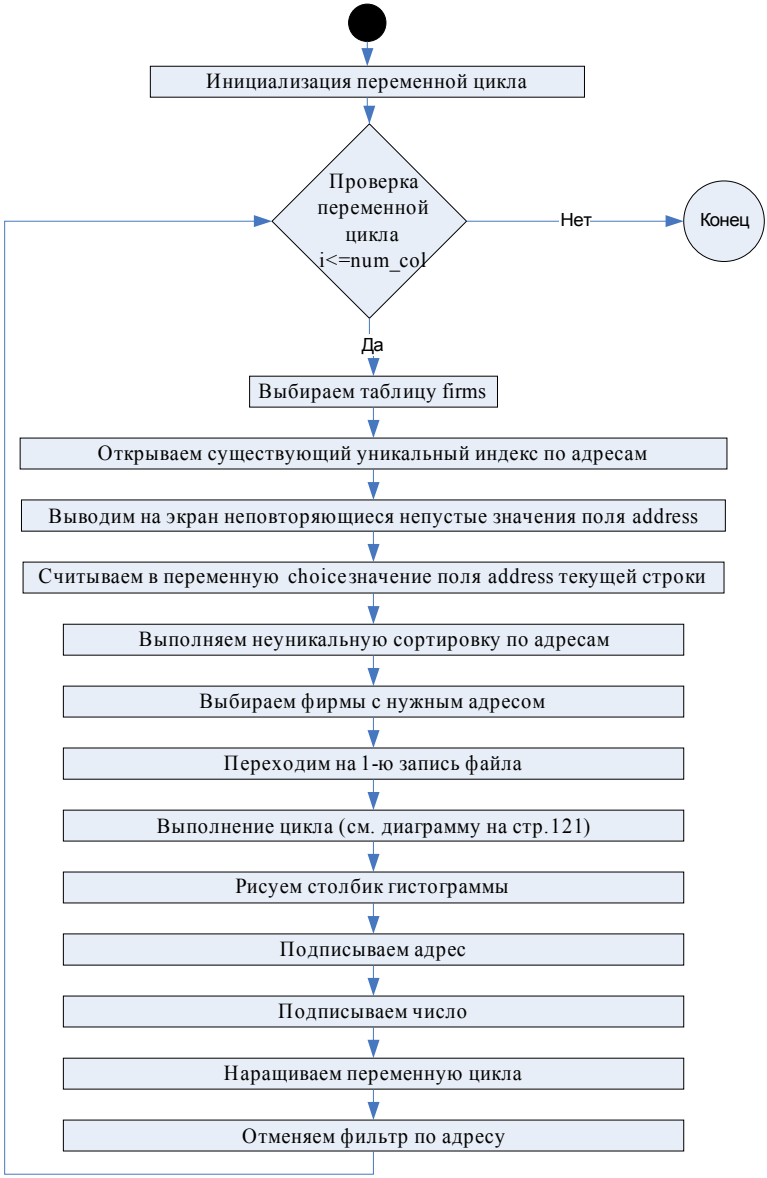
Четвертая нормальная форма (*4НФ*) является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. Весьма не просто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

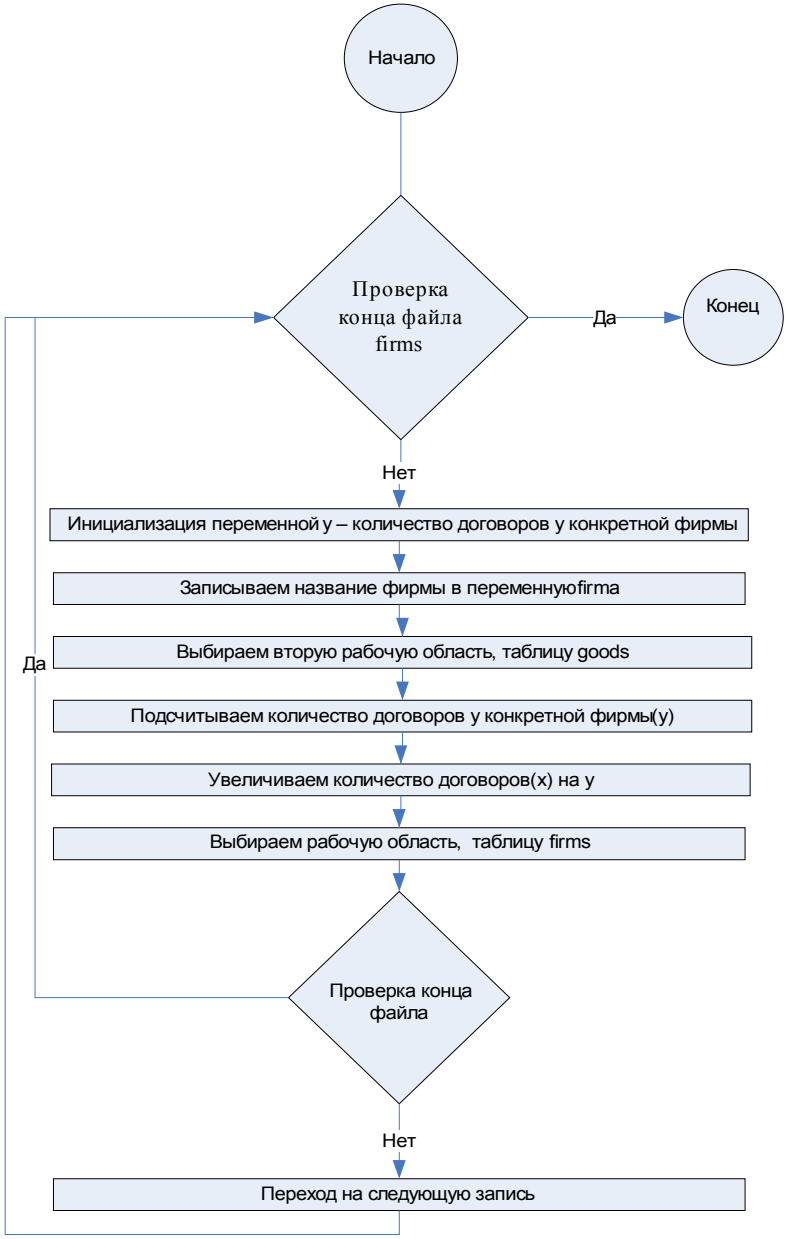
[3] *Полной декомпозицией таблицы* называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Приложение 2
Алгоритмы программ построения гистограмм

Алгоритм программы построения гистограммы сразу по всей таблице (подсчитывается количество товаров по каждому городу, используется две таблицы).

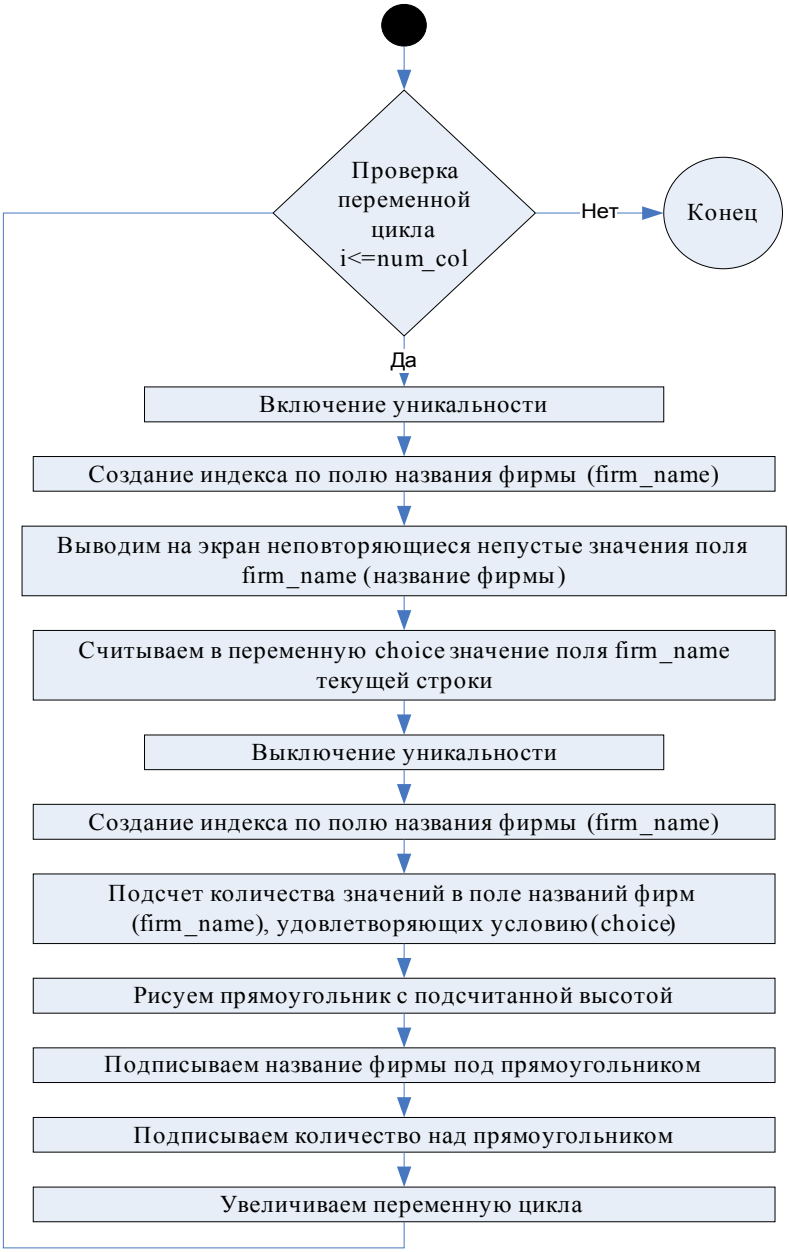



Инициализация переменной цикла
Проверка переменной цикла i<=num_col
Нет
Конец
Да
Выбираем таблицу firms
Открываем существующий уникальный индекс по адресам
Выводим на экран неповторяющиеся непустые значения поля address
Считываем в переменную choiceзначение поля address текущей строки
Выполняем неуникальную сортировку по адресам
Выбираем фирмы с нужным адресом
Переходим на 1-ю запись файла
Выполнение цикла (см. диаграмму на стр.121)
Рисуем столбик гистограммы
Подписываем адрес
Подписываем число
Наращиваем переменную цикла
Отменяем фильтр по адресу


Начало
Проверка конца файла firms
Да
Конец
Нет
Инициализация переменной y – количество договоров у конкретной фирмы
Записываем название фирмы в переменнуюfirma
Выбираем вторую рабочую область, таблицу goods
Подсчитываем количество договоров у конкретной фирмы(y)
Увеличиваем количество договоров(x) на y
Выбираем рабочую область, таблицу firms
Проверка конца файла
Да
Нет
Переход на следующую запись

ПРИЛОЖЕНИЕ 2

Алгоритм программы построения гистограммы по заданному пользователем количеству значений таблицы (подсчитывается количество товаров по каждой фирме, используется одна таблица).


Проверка переменной цикла i<=num_col
Нет
Конец
Да
Включение уникальности
Создание индекса по полю названия фирмы (firm_name)
Выводим на экран неповторяющиеся непустые значения поля firm_name (название фирмы)
Считываем в переменную choice значение поля firm_name текущей строки
Выключение уникальности
Создание индекса по полю названия фирмы (firm_name)
Подсчет количества значений в поле названий фирм (firm_name), удовлетворяющих условию (choice)
Рисуем прямоугольник с подсчитанной высотой
Подписываем название фирмы под прямоугольником
Подписываем количество над прямоугольником
Увеличиваем переменную цикла

Приложение 3
Команды языка FoxPro, используемые в этом пособии

Таблица П3.1

Команда	Описание команды
@ <row,col> GET <memvar/field>	Создание поля ввода
@ <row,col> SAY <memvar/field>	Вывод на экран
? <expr>	Вывод на экран
APPEND [BLANK]	Добавление [пустой]строки
BROWSE [FIELDS …]	Просмотр [определенных полей] таблицы
CLEAR	Очистка текущего окна
COUNT FOR	Подсчет количества строк, удовлетворяющих условию
CREATE	Создание таблицы
CREATE DATABASE	Создание базы данных
CREATE FORM	Создание формы
CREATE LABEL	Создание этикетки
CREATE MENU	Создание меню
CREATE REPORT	Создание отчета
DELETE NEXT 1	Удаляет текущую запись
DELETE FOR	Удаляет записи, удовлетворяющие условию
DELETE TAG	Удаление индексного тега из составного индекса
DROP TABLE	Удаление таблицы
DO CASE...ENDCASE	Выполнение по условию
DO FORM	Запуск формы на исполнение
DO WHILE...ENDDO	Цикл ”пока”
FOR...ENDFOR	Цикл со счетчиком
IF...ENDIF	Логический переход
GOTO	Переход на определенную запись в таблице
LABEL FORM	Запуск этикетки
LOCATE	Поиск информации в индексированной или неиндексированной таблице

Продолжение табл. П3.1

Команда	Описание команды
MODIFY COMMAND	Редактирование программы
MODIFY DATABASE	Редактирование базы данных
MODIFY FORM	Редактирование формы
MODIFY LABEL	Редактирование этикетки
MODIFY MENU	Редактирование меню
MODIFY REPORT	Редактирование отчета
MODIFY STRUCTURE	Редактирование структуры таблицы
PACK	Окончательное удаление записей, помеченных для удаления
QUIT	Выход из Visual FoxPro
RECALL NEXT 1	Снимает с записи метку об удалении
REPLACE	Помещает данные в поля записи таблицы
REPORT FORM	Запуск формы
RUN	Запуск внешней программы
SEEK	Поиск в индексированной таблице
SELECT	Изменение текущей рабочей области
SET FILTER TO	Установка / снятие фильтра
SET SYSMENU TO DEFAULT	Установка стандартного меню Microsoft Visual FoxPro
USE	Открытие таблицы

Приложение 4
Список расширений для именования различных типов файлов в Visual FoxPro

Ниже перечисляются расширения и соответствующие им типы файлов, которые используемые в Visual FoxPro.

Таблица П4.1

Расширение	Тип файла
.act	Файл диаграммы действий мастера документирования
.app	Сгенерированное приложение
.cdx	Файл составного индекса
.dbc	Файл базы данных
.dbf	Файл таблицы
.dct	Memo-файл базы данных
.dcx	Файл индекса базы данных
.dll	Файл библиотеки динамических связей Windows
.err	Файл протокола ошибок компиляции
.exe	Исполняемый файл приложения
.fky	Файл макроопределений
.fll	Файл библиотеки динамических связей Visual FoxPro
.fmt	Файл формата
.fpt	Memo-файл таблицы
.frt	Memo-файл отчета
.frx	Файл отчета
.fxp	Откомпилированный файл программы
.hlp	Файл справки
.idx	Файл простого индекса
.lbt	Memo-файл этикетки
.lbx	Файл этикетки
.lst	Файл списка мастера документирования
.mem	Файл сохранения переменной памяти
.mnt	Memo-файл меню
.mnx	Файл меню
.mpr	Сгенерированный файл меню
.mpx	Откомпилированный файл меню
.ocx	Файл элемента управления OLE

Продолжение табл. П4.1

Расширение	Тип файла
.pjt	Memo-файл проекта
.pjx	Файл проекта
.prg	Файл программы
.qpr	Сгенерированный файл запроса
.qpx	Откомпилированный файл запроса
.sct	Memo-файл формы
.scx	Файл формы
.spr	Файл сгенерированной программы экрана (только для предыдущих версий FoxPro)
.spx	Файл откомпилированной программы экрана (только для предыдущих версий FoxPro)
.tbk	Резервная копия memo-файла
.txt	Текстовый файл
.vct	Memo-файл библиотеки визуальных классов
.vcx	Файл библиотеки визуальных классов
.vue	Файл представления FoxPro 2.x
.win	Файл окна

Приложение 5
Полный синтаксис команд SQL, поддерживаемых СУБД Visual FoxPro

Описание команд SQL, поддерживаемых СУБД Visual FoxPro

Visual FoxPro поддерживает следующие команды SQL:

`ALTER TABLE`

Изменение существующей таблицы. Можно изменить имя, тип поля, количество знаков после запятой, диапазон значений, поддержку проверки на NULL и правила целостности ссылочных данных для каждого поля таблицы.

`CREATE CURSOR`

Создание временной таблицы. Каждое поле во временной таблице определяется именем, типом, количеством знаков после запятой, поддержкой проверки на NULL и правилами целостности ссылочных данных. Эти определения могут быть указаны в командной строке или взяты из массива.

`CREATE SQL VIEW`

Вызывает окно дизайнера представления (View Designer), который позволяет создавать SQL представление.

`CREATE TABLE`

Создание таблицы. Каждое поле новой таблицы определяется именем, типом, количеством знаков после запятой, диапазоном значений, поддержкой проверки на NULL и правилами целостности ссылочных данных. Эти определения могут быть указаны в командной строке или взяты из массива.

`DELETE`

Пометка записей таблицы на удаление

`INSERT`

Добавление новой записи в конец существующей таблицы. Поля новой записи содержат данные, перечисленные в команде INSERT, или указанные в массиве.

SELECT

Определяет критерии запроса и выполняет запрос. Visual FoxPro интерпретирует запрос и выбирает определенные данные из таблицы. Команда SELECT встроена в Visual FoxPro как и всякая другая команда Visual FoxPro.

UPDATE

Обновляет записи в таблице. Записи могут быть обновлены согласно результатам предложения SQL SELECT.

Синтаксис команд

```
CREATE TABLE | DBF TableName1 [NAME LongTableName]
[FREE]
    ( FieldName1 FieldType [( nFieldWidth [, nPreci-
sion] )] [NULL | NOT NULL]
    [CHECK lExpression1 [ERROR cMessageText1]]
    [AUTOINC [NEXTVALUE NextValue [STEP StepValue]]]
[DEFAULT eExpression1]
    [PRIMARY KEY | UNIQUE [COLLATE
cCollateSequence]]
    [REFERENCES TableName2 [TAG TagName1]] [NOCP-
TRANS]
    [, FieldName2 ... ]
    [, PRIMARY KEY eExpression2 TAG TagName2 |,
UNIQUE eExpression3 TAG TagName3
    [COLLATE cCollateSequence]]
    [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
    [COLLATE cCollateSequence]
    REFERENCES TableName3 [TAG TagName5]] [, CHECK
lExpression2 [ERROR cMessageText2]] )
    | FROM ARRAY ArrayName

ALTER TABLE TableName1 ADD | ALTER [COLUMN] Field-
Name1
   FieldType [( nFieldWidth [, nPrecision])] [NULL |
NOT NULL] [CHECK lExpression1 [ERROR cMessageText1]]
   [AUTOINC [NEXTVALUE NextValue [STEP StepValue]]]
[DEFAULT eExpression1]
   [PRIMARY KEY | UNIQUE [COLLATE cCollateSequence]]
```

```
   [REFERENCES TableName2 [TAG TagName1]] [NOCP-
TRANS] [NOVALIDATE]
-or-
ALTER TABLE TableName1 ALTER [COLUMN] FieldName2
[NULL | NOT NULL] [SET DEFAULT eExpression2]
   [SET CHECK lExpression2 [ERROR cMessageText2]]
[ DROP DEFAULT ] [ DROP CHECK ] [ NOVALIDATE ]
-or-
ALTER TABLE TableName1 [DROP [COLUMN] FieldName3]
   [SET CHECK lExpression3 [ERRORcMessageText3]]
[DROP CHECK]
   [ADD PRIMARY KEY eExpression3 [FOR lExpression4]
TAG TagName2
   [COLLATE cCollateSequence]] [DROP PRIMARY KEY]
   [ADD UNIQUE eExpression4 [[FOR lExpression5] TAG
TagName3
   [COLLATE cCollateSequence]]] [DROP UNIQUE TAG
TagName4]
   [ADD FOREIGN KEY [eExpression5] [FOR lExpres-
sion6] TAG TagName4
   [COLLATE cCollateSequence] REFERENCES TableName2
[TAG TagName5]]
   [DROP FOREIGN KEY TAG TagName6 [SAVE]] [RENAME
COLUMN FieldName4 TO FieldName5] [NOVALIDATE]

CREATE [SQL] VIEW [ViewName ] [REMOTE]
[CONNECTION ConnectionName [SHARE] | CONNECTION
DataSourceName]
   [AS SQLSELECTStatement]

SELECT [ALL | DISTINCT] [TOP nExpr [PERCENT]]
[Alias.] Select_Item
   [[AS] Column_Name] [, [Alias.] Select_Item [[AS]
Column_Name] ...]
   FROM [FORCE] [DatabaseName!] Table [[AS]
Local_Alias]
   [ [INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL
[OUTER] JOIN DatabaseName!]
   Table [[AS] Local_Alias] [ON JoinCondition ...]
```

```
      [[INTO Destination] | [TO FILE FileName [ADDI-
TIVE] | TO PRINTER [PROMPT] | TO SCREEN]]
      [PREFERENCE PreferenceName] [NOCONSOLE]
[PLAIN] [NOWAIT]
      [WHERE JoinCondition [AND JoinCondition ...]
[AND | OR FilterCondition [AND | OR
FilterCondition ...]]]
      [GROUP BY GroupColumn [, GroupColumn ...]]
[HAVING FilterCondition] [UNION [ALL] SELECTCommand]
      [ORDER BY Order_Item [ASC | DESC] [,
Order_Item [ASC | DESC] ...]]
```

UPDATE [DatabaseName1!]TableName1 SET Column_Name1 = eExpression1
   [, Column_Name2 = eExpression2 ...]
   WHERE FilterCondition1 [AND | OR FilterCondition2 ...]

INSERT INTO *dbf_name* [(*fname1* [, *fname2*, ...])]
   VALUES (*eExpression1* [, *eExpression2*, ...])

-or-
INSERT INTO *dbf_name* FROM ARRAY *ArrayName* | FROM MEMVAR | FROM NAME *ObjectName*

-or-
INSERT INTO *dbf_name* [(*fname1* [, *fname2*, ...])]
   SELECT [(*fname1* [, *fname2*, ...])] FROM *tablename* WHERE *condition*

DELETE FROM [*DatabaseName!*]*TableName*
   [WHERE *FilterCondition1* [AND | OR *FilterCondi-tion2* ...]]

CREATE CURSOR alias_name
   (fname1 type [(precision [, scale])] [NULL | NOT NULL]
   [CHECK lExpression [ERROR cMessageText]]
   [AUTOINC [NEXTVALUE NextValue [STEP StepValue]]]
   [DEFAULT eExpression] [UNIQUE [COLLATE cCollate-Sequence]]
   [NOCPTRANS] [, fname2 ...])
   | FROM ARRAY ArrayName

Татьяна Владимировна Клецова
Наталья Владимировна Овсянникова
Игорь Вениаминович Прохоров

Базы данных

Лабораторный практикум

Редактор Н.В. Шумакова

Оригинал-макет изготовлен Т.В. Клецовой

Подписано в печать 01.09.2008. Формат 60×84 1/16
Печ. л. 8,25. Уч.-изд. л. 8,25. Тираж 120 экз.
Изд. № 3/44. Заказ №

Московский инженерно-физический институт
(государственный университет),
115409, Москва, Каширское ш., 31.
Типография издательства «ТРОВАНТ», г. Троицк Московской области