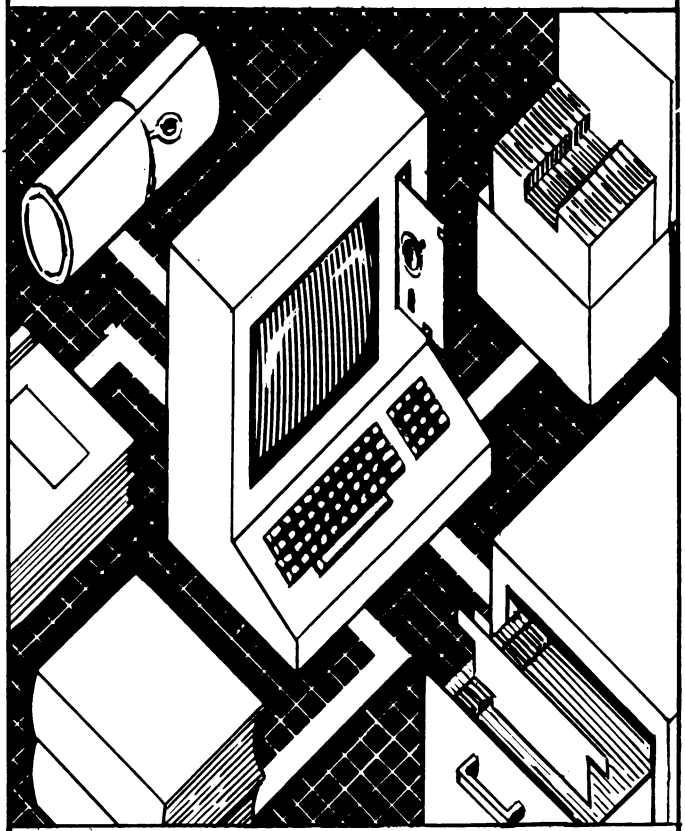
П. Лори

БАЗЫ ДАННЫХ ДЛЯ МИКРОЭВМ

DATABASES

How to manage information on your micro

PETER LAURIE
Southdata Ltd



Designed by Bernard Higton

CHAPMAN AND HALL/METHUEN
London New York

П. Лори

БАЗЫ ДАННЫХ ДЛЯ МИКРОЭВМ

Перевод с английского
Ю.К. Трубина

Москва
«Машиностроение»
1988

ББК 32.973-01
Л78
УДК 681.3.06

Лори П.

Л78 Базы данных для микроЭВМ / Пер. с англ. Ю.К. Трубина. – М.: Машиностроение, 1988. – 136 с.: ил.

ISBN 5-217-00179-8

Книга английского автора представляет собой введение в основы построения баз данных пользователями микроЭВМ, в частности персональных компьютеров. Читатель знакомится с организацией микроЭВМ, типовыми периферийными устройствами, принципами построения баз данных и методами поиска информации. В последней главе рассмотрены перспективы построения и возможности "интеллектуальных" баз данных. Знакомство с книгой не требует опыта работы с вычислительной техникой.

Для инженерно-технических работников всех отраслей промышленности и конторских служащих, использующих микроЭВМ в своей работе.

$Л\frac{2405000000-168}{038(01)-88}$ 168–88

ББК 32.973-01

ISBN 5-217-00179-8 (СССР)
ISBN 0-412-26380-7 (Великобритания)

ОГЛАВЛЕНИЕ

Предисловие 7

Глава 1. ЭЛЕКТРОННО-ВЫЧИСЛИТЕЛЬНАЯ МАШИНА 9
1.1. Интроспективная пишущая машинка 9
1.2. Дисковая память 11
1.3. Операционная система 14
1.4. Периферийные устройства 17
1.5. Клавиатура 22
1.6. Многопультовые системы 26
1.7. Виртуальная память 29
1.8. Как взаимодействуют ЭВМ 29
1.9. Современные устройства внешней памяти ЭВМ 31
1.10. Ошибки в программах 36
1.11. Затраты на программное обеспечение 38
1.12. Сбои и отказы ЭВМ 39

Глава 2. СИСТЕМА УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ 40
2.1. "Разумный" ящик для хранения картотеки 41
2.2. Индексирование 47
2.3. Хеширование 49
2.4. Последовательный индекс 52
2.5. *B*–деревья 54
2.6. Сортировка 59

Глава 3. ИНТЕРФЕЙС КОНЕЧНОГО ПОЛЬЗОВАТЕЛЯ 61
3.1. Формы и их заполнение 61
3.2. Сохранность информации в ЭВМ 64
3.3. Время реакции системы 66
3.4. Поиск в базе данных 68
3.5. Удаление записей 72
3.6. Генератор отчетов 72
3.7. Программирование с использованием базы данных 74
3.8. Базы данных для хранения текстов 75
3.9. Базы данных для хранения графических изображений 78
3.10. Программы расчета рабочих таблиц 80
3.11. Машинная графика 82

Глава 4. МНОГОФАЙЛОВЫЕ БАЗЫ ДАННЫХ 82
4.1. Реляционная алгебра 89
4.2. Иерархическая база данных 95
4.3. Отображение связанных записей на экране 97
4.4. Корректировка данных 100

Глава 5. ЯЗЫКИ ЗАПРОСОВ 101
5.1. Языки программирования 103
5.2. Программирование в системе *Superfile* 105
5.3. Наглядность программ 108
5.4. Логические схемы и словари данных 110
5.5. Язык Пролог 111
Глава 6. ИСТОЧНИКИ И СТОИМОСТЬ ИНФОРМАЦИИ 113
6.1. Распределение информации 116
6.2. Обработка информации 119
6.3. Достоверность информации 122
6.4. Законодательство о защите информации 126
6.5. Забытое прошлое 126
Глава 7. БАЗЫ ДАННЫХ, ОРИЕНТИРОВАННЫЕ НА ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ 128
7.1. Проблемы информации 133
Список литературы 135

ПРЕДИСЛОВИЕ

Эту книгу следует рассматривать как введение в системы управления базами данных для микроЭВМ. Некоторым читателям тема книги может показаться слишком специальной. Однако практически в каждом учреждении, в каждом доме имеются каталоги и картотеки. Если информацию, которую они содержат, хранить в памяти ЭВМ, то ее можно рассматривать как базу данных. Специалисты утверждают, что через пять лет каждый инженерно-технический работник будет иметь доступ к персональной ЭВМ и, следовательно, должен понимать, как работает база данных.

Любители, увлекающиеся вычислительной техникой в свободное время, найдут в этой книге много полезного. Рост производства и расширение областей применения микроЭВМ происходят столь стремительно, что наши знания о них не успевают пополняться. Вот почему все больший акцент делается на "удобное для пользователя" программное обеспечение, призванное работать без какого-либо вмешательства со стороны пользователя. Большой выбор пакетов программ управления базами данных, ориентированных на неподготовленного пользователя, способствует, на мой взгляд, тому, что многие проблемы организации информации оказываются скрытыми. Усилия упорно направляются на разработку программ, заставляющих наивных потенциальных покупателей ЭВМ поверить в то, что они не должны понимать, что происходит внутри машины. Можно провести аналогию между использованием микроЭВМ сегодня и легковых автомобилей в начале этого века. Отправляясь в далекое путешествие на автомобиле, водитель где-то на полпути уже мог самостоятельно отшлифовать гильзы цилиндров. Если Вы становились автолюбителем, то осваивали профессию автомеханика. Точно так же, если Вы не понимаете, как работает база данных в начале ее создания, то к концу разработки Вы, безусловно, будете знать об этом. Как повторял своим солдатам Суворов: "Тяжело в учении – легко в бою".

Лица, профессионально занимающиеся вычислительной техникой, прочитают эту книгу с интересом. До недавнего времени разработка и внедрение систем управления базами данных считались уделом небольшой

группы специалистов, которые создали на высоком уровне абстракции теорию баз данных. Часто эта теория выглядит совершенно оторванной от практических потребностей пользователей. В 1980 г., когда фирма "Саутдейта" начала разрабатывать пакет программ управления базой данных*, я с удивлением обнаружил, как мало полезного можно было найти в опубликованных на эту тему работах.

Мы писали программы в темноте невежества, и это пошло нам на пользу (см. гл. 4). Я понял, каким требованиям должны удовлетворять базы данных только после того, как в течение трех лет побеседовал с тысячами людей, которые хотели организовать хранение данных на микроЭВМ. Эти встречи породили бесконечные споры с программистами, в том числе моим сыном Бенеттом, без участия которого не было бы ни фирмы "Саутдейта", ни пакета программ *Superfile*, ни этой книги. Только после длительной напряженной работы над программами, после многих попыток получить результаты, удовлетворяющие реальным практическим приложениям, мы, наконец, создали работающую систему. Эта система стала основой для книги, которая, как я надеюсь, раскроет читателю некоторые проблемы создания баз данных.

Заранее прошу извинить меня за то, что в книге описана собственная разработка. Автор не должен преследовать коммерческие цели. Но мир ЭВМ так молод и невелик, и так быстро развивается, что каждый, у кого есть идеи, старается "застолбить" их различными способами. Складывается ситуация, напоминающая времена заселения дикого Запада, когда шериф, преследовавший других за кражу скота, владел самым большим ранчо в округе. К сожалению, среди тех, кто занимается вычислительной техникой, нет незаинтересованных наблюдателей.

Конечно, было бы неплохо описать в книге особенности и возможности всех существующих пакетов программ управления базами данных, но таких пакетов разработано слишком много, только в Англии их насчитывается около 60, а в США – более 150. Чтобы подробно описать одну систему управления базой данных (СУБД) требуется месяц напряженного труда. Чтобы описать все имеющиеся сегодня СУБД для микроЭВМ, потребуется 17 лет. За такой срок все разработки устареют.

Эта книга не претендует на то, чтобы стать справочником для программистов-профессионалов. Достаточно взглянуть на "библию программистов" Дональда Кнута "Искусство программирования" в семи томах [3], чтобы понять, что ни одна книга меньшего объема, не может стать полезным пособием. Цель книги: довести основные проблемы и их решения до неподготовленных пользователей и в то же время помочь несколько озадаченным профессионалам сделать правильный выбор среди множества конкурирующих друг с другом программных продуктов.

---

* Пакет программ известен в Англии под названием *Superfile* (Суперфайл), в США он называется *Avatar*.

ГЛАВА 1
ЭЛЕКТРОННО-ВЫЧИСЛИТЕЛЬНАЯ МАШИНА

База данных представляет собой совокупность массивов взаимосвязанной информации, хранящуюся в памяти ЭВМ. С ее помощью можно получить доступ к данным не только по номеру дела или фамилии адресата, как это имеет место в обычной справочной картотеке, но и по любому другому признаку.

Прежде чем рассматривать вопросы создания базы данных, ее преимущества и области применения,не мешает сделать отступление от основной темы и описать отличительные особенности самой ЭВМ и влияние ее технической характеристики на разработку базы данных.

1.1. ИНТРОСПЕКТИВНАЯ ПИШУЩАЯ МАШИНКА

Об ЭВМ можно писать много. Однако в рамках этой книги, где мы рассматриваем применение ЭВМ лишь для хранения и поиска информации в базах данных, ее можно сравнить с обычной электрической пишущей машинкой, обладающей некоторой памятью, и объединенной с ящиком для хранения картотеки. Под памятью я имею в виду способность машины просматривать, что она напечатала не только накануне, но и два года назад, и выполнять запросы вроде : "Найдите сведения о человеке по имени Том" или "Составьте список лиц, чья задолженность превысила 500 фунтов стерлингов".

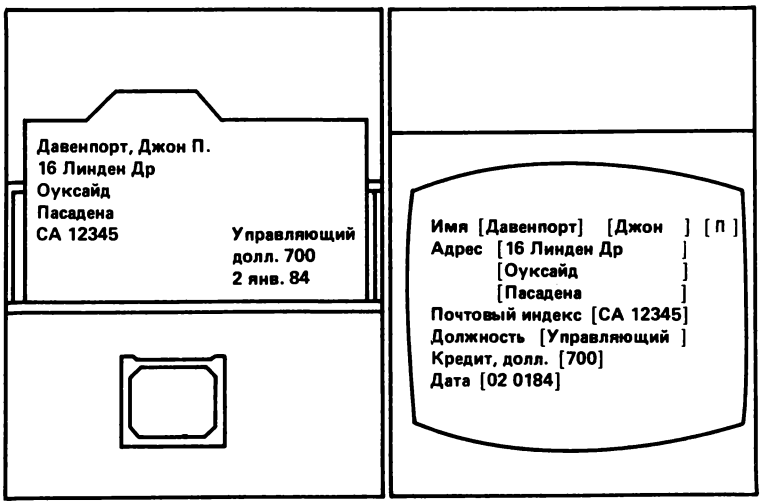


Рис. 1. Запись в базе данных (справа) не отличается от записи на карточке в картотеке (слева). Каждый элемент информации на карточке помещается в именованное поле данных на экране ЭВМ

Имя	[Джон] [П] [Давенпорт]	Фамилия
Адрес	[16 Линден Др]	
Адрес	[Оуксайд]	
Адрес	[Пасадена]	
Штат	[Калифорния]	
Почтовый индекс	[СА 12345]	
	23 октября 1984 г.	Дата
	Уважаемый г–н [Давенпорт]	
	[Это пример того, как деловое письмо можно представить в виде записи базы данных]	
	Благодарю Вас, искренне Ваш	
Подпись	[П. Лори]	

Рис. 2. Элементы, из которых состоит деловое письмо, легко разместить в полях записи базы данных

Пока представим себе (в гл. 4 многие вопросы будут рассмотрены подробно), что информация в базе данных состоит из ряда одинаковых по форме "записей". Запись включает группу взаимосвязанных полей информации, каждое из которых представляет собой последовательное число байтов, отведенное для хранения значений данных определенного типа: фамилии, даты, цены товара и т. п. Деловое письмо, хранящееся в картотеке, можно представить в виде записи данных. В последующих главах подробно рассмотрены способы организации записи.

База данных включает многие массивы информации, размещенные на диске и содержащие записи с идентичной организацией. Условия, в которых работают системы баз данных, характеризуются значительными требованиями к возможностям памяти ЭВМ, вот почему основные параметры устройств памяти являются для нас очень важными.

Машина имеет два вида устройств памяти: модули оперативной памяти с произвольной выборкой (оперативные запоминающие устройства – ОЗУ), непосредственно связанные с процессором, и устройства внешней памяти на магнитных дисках. Эти устройства, которые могут быть встроены в основной блок машины, функционально с ним не связаны, и являются по отношению к нему внешними, так же как клавиатура и видеоконтрольное устройство (дисплей).

Емкость ОЗУ в настольных микроЭВМ относительно невелика и может изменяться в пределах от 64 до 500К байт*. Когда отключается питание, содержимое ОЗУ стирается. Накопители на магнитных дисках предназначены для долговременного хранения больших объемов информации. В последних моделях микроЭВМ емкость накопителя на гибких магнитных дисках составляет 1,5М байт, а на жестких (твердых) дисках –

*1К байт = 1024 байт, 1М байт = 1024^2 байт – *Прим. пер.*

20М байт. Если у Вас есть финансовая возможность, Вы можете купить накопители на тысячи мегабайт дисковой памяти. Далее будут рассмотрены устройства, которые появятся в ближайшем будущем и смогут хранить огромные количества информации. Их разработка ведется на основе новой дешевой технологии.

Напомним, что слово в тексте в среднем состоит из 6 букв*, поэтому память в 1М байт содержит 166666 слов или вмещает три средних по размеру повести. Квалифицированная машинистка печатает со скоростью 80 слов в минуту и ей потребуется работать без перерыва 33 ч,чтобы ввести в ЭВМ 1М байт информации. Удивительно, но такой большой памяти быстро начинает не хватать.

ЭВМ, с помощью которой я писал эту книгу, имела 16М байт дисковой памяти, и 15 из них были заполнены к концу работы. Мы можем удалить файлы, которые кажутся нам больше ненужными, но всегда оказывается, что они очень нужны кому-то другому. Большинство дорогих твердых дисков на 3/4 заполнены файлами, о существовании которых все давно забыли. Вариант закона Паркинсона: данные накапливаются, чтобы заполнить имеющуюся память – действует безотказно. Новые накопители на магнитных дисках большой емкости смогут вместить результаты работы квалифицированной машинистки в течение всей ее жизни. Однако не пройдет и года,как эти накопители будут заполнены.

ЭВМ не может непосредственно манипулировать с информацией, хранящейся на диске. Для того чтобы произвести расчеты или поиск информации, данные необходимо переписать в оперативную память. В этом смысле можно провести аналогию между оперативной памятью и рабочей поверхностью письменного стола. Вы достаете папки с бумагами из ящиков и шкафов, выкладываете необходимые Вам документы на стол и работаете с ними, после чего возвращаете их на место. Однако если Вы приобрели пакет программ управления базами данных, то у Вас появляется расторопный и сообразительный помощник, который найдет интересующие Вас бумаги, руководствуясь Вашими туманными намеками относительно их содержания.

1.2. ДИСКОВАЯ ПАМЯТЬ

Информация хранится на поверхности диска, независимо от того гибкий он или твердый, в виде очень малых намагниченных участков. Для записи 0 используется одна ориентация магнитных силовых линий на участке, для записи 1 ориентация линий меняется на противоположную. (В действительности все, как обычно, сложнее, но в первом приближении можно ограничиться таким представлением физической записи). Двоичное число 0 или 1 называется битом.

Поверхность диска покрывается слоем магнитного материала, в ко-

* Имеются в виду слова английского языка. – *Прим. пер.*

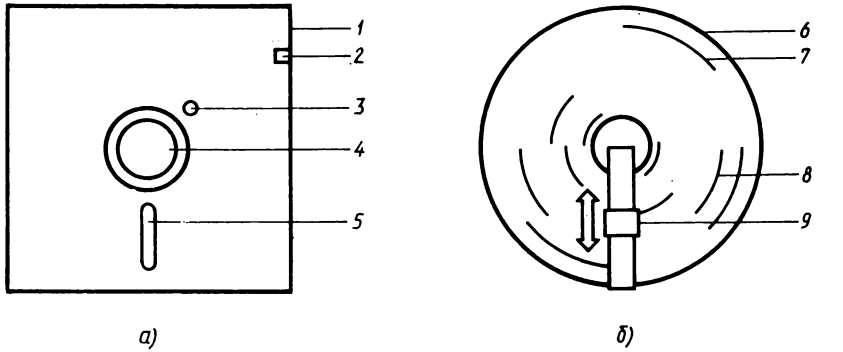


Рис. 3. Гибкий диск (б) вращается внутри специального конверта (а). Тем самым обеспечивается доступ головки чтения–записи к информации, записанной на диске. Жесткий диск типа "Винчестер" работает аналогичным образом, но его нельзя снимать с дисковода:

1 – конверт; *2* – вырез защиты записи; *3* – синхродорожка; *4* – отверстие для вала гибкого магнитного диска; *5* – прорезь для головки чтения–записи; *6* – диск; *7* – дорожка данных (данные записываются на дорожку блоками в пределах секторов, на которые разбивается дорожка; блоки пишутся на любое свободное место на диске); *8* – дорожка данных; *9* – головка чтения–записи

тором головка чтения–записи создает намагниченные участки. Сама головка представляет собой миниатюрный электромагнит, перемещаемый по поверхности диска. Принцип записи информации на диске ничем не отличается от записи с помощью магнитофона, только там магнитная лента протягивается вдоль неподвижной головки, а здесь и головка и диск движутся. Одна и та же головка используется для считывания и записи информации.

Диск вращается, а магнитная головка выдвигается вперед или назад. В результате сложения этих двух движений магнитная головка может записать информацию на любую часть поверхности диска в виде концентрических "дорожек", каждая из которых подразделяется на "секторы".

Для того чтобы установить головку на требуемую дорожку и дождаться, когда подойдет нужный сектор, необходимо некоторое время, примерно 0,2 с для накопителя с гибким магнитным диском и около 0,003 с для накопителя с жестким диском.

При оценке быстродействия программ управления базами данных следует подсчитать число блоков данных, которые должны быть прочитаны в основную память или записаны из нее.

Многим читателям известно, что биты, представленные на диске крошечными намагниченными участками, объединяются в байты. Каждый байт состоит из 8 битов. Поскольку бит может иметь два значения: либо 0, либо 1, байт способен принимать $2^8 = 256$ различных значений, которых с избытком хватает, чтобы закодировать прописные и строчные буквы

алфавита, цифры, знаки пунктуации, специальные знаки и все другие символы, которые входят в набор символов ЭВМ.

Файл содержит последовательность байтов и ничего больше. ЭВМ может интерпретировать байты как команды для процессора (в этом случае файл называется программным), как букву или цифру или иначе.

Поскольку базы данных хранят информацию, относящуюся к конкретным областям человеческой деятельности, для нас важно рассматривать байт как отдельный символ, т. е. букву или цифру, которой соответствует определенная клавиша на клавиатуре ЭВМ.

В качестве кода ЭВМ широко распространен американский стандартный код для обмена информацией (*ASCII*)*, по которому каждый символ клавиатуры кодируется десятичной цифрой от 0 до 127. Прописные и строчные буквы имеют свои коды и рассматриваются как различные символы. Код буквы A в *ASCII* равен 65, код a равен 97. Имеется ряд непечатаемых знаков вроде "Подача на одну строку". "Перевод каретки" и т. д., а также 32 знака, вводимых одновременным нажатием клавиши управления и какой-либо буквы. Эти знаки используются для ввода команд. Например, при обработке текста нажатие клавиши "Управление" и буквы F вызывает перемещение курсора вперед на одну позицию.

Система кода *ASCII* хорошо работает с набором букв английского алфавита, однако ее применение для других языков вызывает ряд трудностей. Связаны они с кодированием знаков специальных для каждого языка, например é в испанском языке или ç во французском. *ASCII* выделяет коды для таких букв, но предполагается, что пользователь использует наборы букв испанского или французского языка, и никогда не использует оба набора вместе. Например, в США кодом 43 кодируется символ "/". В Англии этот же код означает знак фунта стерлингов. Если в Англии Вы захотите включить в набор символ ЭВМ "/", Вам придется отказаться от знака фунта стерлингов. Обратное утверждение справедливо для американцев. Код, используемый для кодирования символа] в Англии и США, в Швеции соответствует букве ä, а в ФРГ и ГДР букве ü. В результате Вы не можете хранить в базе данных информацию на двух и более европейских языках. Это создает определенные трудности. Несомненно, американские специалисты, которые разрабатывали *ASCII*, исходили из того, что знание иностранных языков никому не нужно.

Впервые мое внимание к этой проблеме привлек один англичанин, который хотел вести базу данных о военных публикациях, поступавших к нему изо всех стран. Он быстро понял, что создание автоматической библиографической системы не позволит правильно записывать фамилии авторов публикаций. Можно не сомневаться, что дальнейшее бурное развитие микропроцессорной техники очистит европейские языки от их своеобразных знаков.

Если Вы используете символы букв арабского алфавита, Вы в состоянии закодировать каждый символ одним байтом. Для знаков китайского

*В нашей стране применяется аналогичный код КОИ-7. – *Прим. пер.*

или японского языка Вы можете выделить по два байта и таким образом закодировать $256^2 = 65536$ знаков.

Однако, как правило, мы хотим хранить в базе данных буквы и цифры родного языка и байт, как элемент данных, вполне подходит для этих целей.

ЭВМ хранит байты в файле на диске. Средства обработки файлов в операционных системах микроЭВМ, таких как *MS-DOS* и *CP/M*, рассматривают файл вполне определенным образом: как длинную последовательность байтов. Файл имеет имя и длину. Система управления базой данных хранит необходимую ей информацию в одном или нескольких файлах. Как правило, она создает отдельные файлы индексов к данным, но может хранить в отдельных файлах и другую информацию.

ЭВМ хранит в виде файлов не только данные, но и программы, а программы используют свои файлы. Если по Вашей директиве ЭВМ распечатает каталог файлов, хранимых на диске, Вы увидите длинный список имен файлов в форме *xxxxxxx.yyy*. Часть, обозначенная символами *xxx*, представляет собой собственное имя файла, в то время как символы *yyy* обозначают расширение имени файла. Расширение имени файла указывает пользователю и ЭВМ на тип файла. Программные файлы обычно имеют расширение *COM* или *CMD*, так как они содержат управляющие команды. К имени текстовых файлов часто добавляются *TXT*, к имени файлов базы данных – *DAT* и т. д. Используя расширение имени, Вы можете заставить машину отобразить на экране, скопировать, вывести на печать или выполнить какую-либо другую операцию со всеми файлами одного типа. Например, главы этой книги хранились в памяти машины как набор файлов с именами C1.B, C2.B, C3.B, ... C8.B*. Расширение .B позволяло мне копировать целиком всю книгу с помощью единственной команды COPY*.BB: (копировать все файлы с расширением .B, используя дисковод B).

Система управления базами данных использует для работы с файлами программы операционной системы. Операционная система – это программное обеспечение, которое управляет техническими средствами и выполняет все необходимые действия по их обслуживанию: например, считывает в память или записывает из нее информацию на диск, вводит символы с клавиатуры, выводят данные на экран или печатающее устройство.

1.3. ОПЕРАЦИОННАЯ СИСТЕМА

Средства операционной системы осуществляют важную и незаметную для внешнего наблюдателя работу по управлению самой ЭВМ. Они выполняют сложные функции. После того как Вы приобрели и запустили новую микроЭВМ, все выглядит очень просто. Вы записываете свой пер-

* C – начальная буква английского слова *chapter*, означающего "глава". – *Прим. пер.*

вый файл на новый, совершенно чистый диск, и сразу за ним начинаете писать второй файл, затем третий и т. д., пока диск не заполнится информацией. Использовано все пространство дисковой памяти. Но жизнь течет, и вскоре Вы вынуждены внести изменения в первый файл. Теперь он будет либо короче, либо длиннее первоначального. Если короче, то на диске появится неиспользованное пространство, если длиннее, то где найти дополнительное место? Вы не можете продолжить первый файл, так как сразу за ним начинается второй файл.

Выход заключается в следующем: диск делится на небольшие блоки длиной от 128 до 1024 байт. В каждом блоке хранится часть файла (часто блоки ошибочно называют записями). Мы будем писать слово "запись" с маленькой буквы, когда речь идет о блоке данных, и с большой буквы, когда имеются в виду записи базы данных. Последние мы будем рассматривать в следующей главе. Когда файл уничтожается или становится короче, часть блоков на диске высвобождается и может быть использована для других файлов, которые, наоборот, пополняются. Операционная система ведет свой собственный указатель, в котором хранится информация о номерах блоков, используемых каждым файлом, и в какой последовательности они следуют. Когда ЭВМ выполняет по программе команду чтения файла, операционная система автоматически передает записи с диска в оперативную память в правильной последовательности. Если программой предусматривается чтение только части файла, она передает операционной системе номер записи и имя файла. Если в данный момент работает программное обеспечение базы данных, то оно определит номер соответствующей записи по своим индексам (см. гл. 2). Операционная система затем передает запись в буфер памяти ЭВМ, где запись станет доступной для программы управления базой данных. Буфер представляет собой часть ОЗУ, которую программы используют так же, как мы используем доски объявлений, помещая на них сообщения для других лиц. Вообще говоря, чем больше размеры буфера, тем лучше работает система, поскольку уменьшается количество доступов к физическим блокам, необходимое для выполнения той или иной операции. Однако, выделяя больше ячеек памяти под буферы, мы уменьшаем память, предназначенную для хранения программ. Поэтому разработчикам баз данных приходится находить компромиссные решения.

Программы работают в "операционной среде". Наиболее широкое распространение получили операционные системы: *CP/M*80 (для восьмиразрядных машин), *CP/M*86 (для шестнадцатиразрядных машин), *MS-DOS, Unix, Idris* и *Xenix* (только для шестнадцатиразрядных машин). Программное обеспечение СУБД использует средства операционной системы для обработки своих файлов. С помощью этих средств можно получить доступ ко всему файлу или его отдельной записи. Система управления базой данных определяет номер записи, просматривая свои индексные файлы.

Очень важной характеристикой программного аппарата базы данных является среднее время обработки запросов в системе. Разработчики пакетов программ управления базами данных изо всех сил стараются

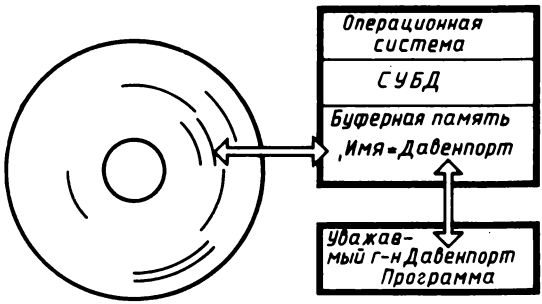


Рис. 4. Информация с диска считывается операционной системой и передается системе управления базой данных, после чего она становится доступной прикладной программе, в данном случае программе печати типового письма

снизить это время. Заказчикам и будущим пользователям таких систем полезно знать, какие трудности приходится при этом преодолевать программистам. Распределение записей между дорожками и секторами диска — это функция программ операционной системы, которые пишут системные программисты.

Для того чтобы операционная система могла найти конкретный элемент информации, ей надо сообщить номер записи и имя файла. Операционная система просматривает указатель, определяет физический адрес записи на диске и устанавливает головку над требуемой дорожкой. Затем программа ждет, пока диск повернется на некоторый угол и под головкой окажется требуемый сектор, после чего происходит считывание информации с диска в буфер оперативной памяти ЭВМ. Система управления базой данных переписывает информацию в свои собственные буферы и может запросить следующий блок, если запись базы данных размещается в нескольких записях на диске (как это чаще всего бывает).

Объем информации, хранимой в базе данных, зависит от емкости накопителей на магнитных дисках, на которых размещаются файлы системы.

Размеры дисковой памяти определяются физическими размерами дисков, основным ограничением здесь является стоимость устройств, и максимальной емкостью накопителей, с которыми может работать операционная система. Например, операционная система *CP/M* для восьмиразрядных машин позволяет использовать только диски 8М байт.

Реальные объемы хранящейся в базе данных информации будут наверняка меньше, так как СУБД поддерживает индексные файлы, и может использовать только записи постоянной длины, что приводит к неэкономному использованию памяти.

1.4. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

Внешне ЭВМ ничем не отличается от ящика – черного ящика, если Вы купили машину черного цвета, или ящика другого цвета. Оформление зависит от фирмы-изготовителя. (Некоторые люди полагают, что форма и цвет ЭВМ имеют какое-то значение. Они могут восторгаться внешним видом машины, не понимая, почему программисты, глядя на ту же машину, обмениваются презрительными взглядами).

При работе ящик ничем себя не проявляет. В лучшем случае на передней панели Вы увидите, как загораются одна или две лампочки. Вы можете получить пользу от ЭВМ только в том случае, если присоедините к ней периферийные устройства: клавиатуру и экран (вместе их обычно называют терминалом), печатающее устройство и (гораздо реже) другие устройства, например графопостроитель. Большинство настольных микроЭВМ укомплектованы клавиатурой и экраном, но если Вы купили более мощную многопультовую микроЭВМ, Вам придется приобрести отдельно видеоконтрольные устройства (дисплеи).

Раньше Вы покупали у фирмы–изготовителя ЭВМ автоматизированную систему "под ключ". Фирма за большие деньги устанавливала ЭВМ и запускала на ней программы, которые делали то, что Вы хотели. Больше Вы ни о чем не беспокоились. Теперь, когда появились недорогие персональные ЭВМ, и каждый стал сам по себе инженером-электронщиком и системотехником, возникла проблема. Она заключается в том, что большей частью Вы будете использовать готовые пакеты программ, авторы которых не в состоянии предугадать, какие конкретные модели печатающих устройств и дисков Вы выберете из тысяч моделей, предлагаемых на рынке. Разработчики программ вынуждены включать в них настраиваемые модули, которые позволяют адаптировать программное обеспечение к Вашей машине.

К сожалению, наибольшие трудности связаны именно с начальным запуском ЭВМ или программ, когда знания о них минимальны. Даже опытные программисты вынуждены иногда работать до седьмого пота, чтобы освоить готовый пакет программ.

На эту тему можно написать целую книгу, но важно запомнить следующее:

1) для каждого видеоконтрольного устройства существуют свои коды, управляющие перемещением курсора в пределах рабочего поля экрана. Иногда достаточно сообщить программе тип дисплея, по которому она определит необходимые коды. Это бывает в тех счастливых случаях, когда разработчики пакета программ учитывают особенности наиболее распространенного периферийного оборудования. Чаще Вам необходимо ввести в систему различные коды, которые Вы можете найти (правда не всегда) в руководстве по эксплуатации терминала;

2) печатающее устройство также имеет свои особенности. В некоторых моделях выбор требуемого вида печати осуществляется с помощью

HELP press F1	FIND the first record which contains the text enered in any field. ENTER TEXT	COMMANDS press F3
PROMPTS press F2	(CONTINUE finds further records)	ESCAPE press ESC

COUNTRIES OF THE WORLD

Country: INDONESIA — Continent: S.E.ASIA

Capital: DJAKARTA — Currency: RUPIA

Languages: BAHASIA INDONESIA

----------DATA----------

Population	135.4	millions
Land Area	1919	thousards of square kilometres
Gross Domestic Product	155	U.S. dollars per capita

>find
"indonesia"
>find
"china

Рис. 5. В программное обеспечение даже самых маленьких микроЭВМ теперь включают пакеты управления базами данных. Например, фирма "Синклер" снабжает таким пакетом свою микроЭВМ, хотя она имеет устройство внешней памяти небольшой емкости на магнитном носителе, что не позволяет использовать обычные базы данных. Ниже показан фрагмент базы данных для микроЭВМ, в которой накапливается социально-экономическая информация о странах мира

Рис. 6. Минимальная конфигурация ЭВМ для работы с базой данных включает накопитель на гибких магнитных дисках. Показано также шестиклавишное устройство ввода информации

последовательности кодов команд, высылаемых процессором. Устройства печати могут использовать расширенные и нормальные знаки, например М шире, чем I, или все знаки могут быть одной ширины, как в наборе символов пишущей машинки. Если Вы хотите оформить результаты расчетов в виде таблиц, Вам не нужен расширенный шрифт, поскольку столбцы таблицы окажутся невыравненными. Кроме того, необходимо сообщить печатающему устройству количество знаков в строке и длину страницы (обычно это 80 или 132 символа и 66 строк в странице). Некоторые печатающие устройства автоматически продвигают бумагу на один интервал в конце каждой строки, переданной процессором, другие этого не делают. Программе надо сообщить, выполняется эта команда или отменяется.

Если Ваше печатающее устройство может подчеркивать текст, печатать шрифт или индексы, то программные средства должны "знать" коды команд, которые эти возможности реализуют. Многие пакеты про-

Рис. 7. Персональный компьютер PC XT фирмы ИБМ предназначен для решения широкого круга задач. В составе программного обеспечения ЭВМ – несколько пакетов программ управления базами данных

грамм заранее учитывают особенности наиболее часто встречающихся печатающих устройств, и может быть Вам повезет в этом отношении;

3) наконец, надо разобраться с портами ввода-вывода. Порт – это многоразрядный вход или выход из ЭВМ, через который данные принимаются или передаются в терминал или печатающее устройство.

Большинство современных микроЭВМ имеют, по крайней мере, два последовательных и один параллельный порт. Они будут находиться под управлением операционной системы, и каким-то образом надо подсказать ей, какой порт Вы используете.

В операционной системе *MS-DOS* главный последовательный порт обозначается *COM-1*, в *CP/M*86, – *LST*, в *Unix* он носит название */dev/lp*. Для последовательного порта необходимо запрограммировать скорость посылки данных в периферийное устройство или из него. Данные можно передавать с контрольным битом четности или нечетности; с 1, 1,5 или 2 стоповыми битами. Количество битов в байте для порта может равняться 7 или 8. Часто эти сведения отсутствуют в руководствах. Иногда о них не знают даже изготовители систем. За пять лет работы в области вычислительной техники я ни разу не видел, чтобы печатающее устройство запустили меньше чем за 2 ч. В худшем случае на это может уйти год.

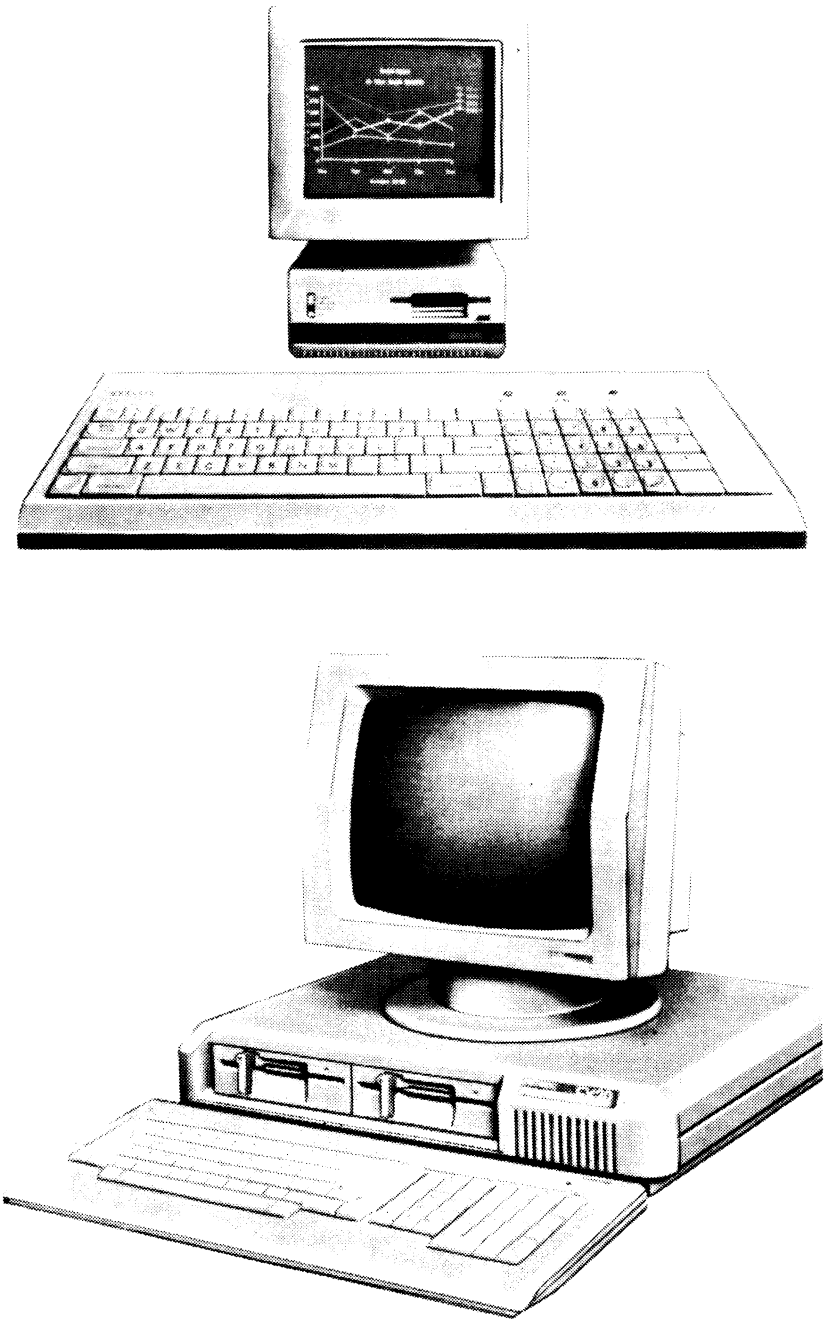


Рис. 8. МикроЭВМ "Фьюче Компьютерс" (вверху) и "Априкотс" (внизу) могут работать в сетях ЭВМ, обеспечивая нескольким пользователям доступ к одной и той же базе данных

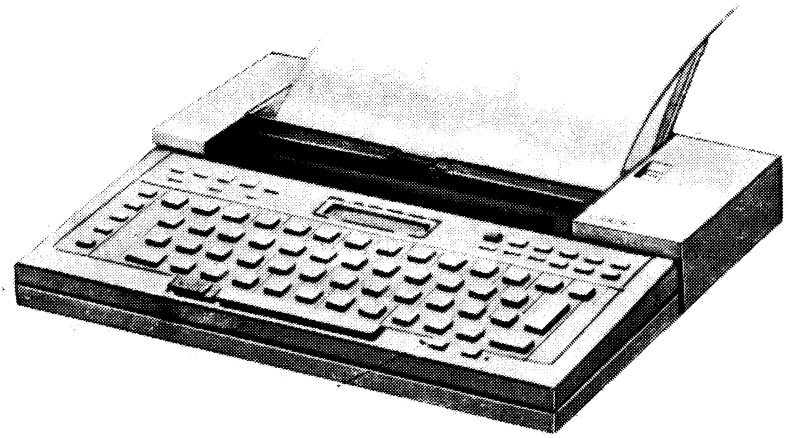


Рис. 9. Портативная ЭВМ дает возможность руководителю использовать вычислительную технику в деловых поездках для обработки текстов и вычислений. При необходимости он может подключаться к базе данных на большой ЭВМ по телефонному каналу связи

1.5. КЛАВИАТУРА

В состав каждой микроЭВМ обязательно входят одно или более периферийных устройств стоимостью 1500 фунтов стерлингов и более. В мире ежегодно производятся миллионы компьютеров и фирмы – изготовители периферийного оборудования заинтересованы в создании новых устройств. Более того, рынок микроЭВМ только формируется и многие потребители не обладают еще достаточными знаниями об их устройстве и не понимают, что характеристики компьютера определяются в первую очередь возможностями процессора, оперативной памяти и программного обеспечения, а не периферийного оборудования. Промышленность постоянно навязывает покупателям новые модели периферийных устройств. Возможно, некоторые из них имеют определенные преимущества по сравнению с ранее выпущенными устройствами. Например, "Микрорайтер" – электронный вариант стенографической машинки, используемой в американском судопроизводстве. Это устройство ввода информации одной рукой. На нем имеется по клавише для каждого пальца плюс еще две дополнительные клавиши. Каждой букве соответствует определенная комбинация нажатых клавиш. Разработчики устройства утверждают, что оно позволяет быстрее вводить данные, делать меньше ошибок, работа на нем

Рис. 10. Фирма "Эппл" выступила пионером в области отображения информации на экране ЭВМ. Вместо того, чтобы вводить имена программ и команды с клавиатуры, пользователь новой модели микроЭВМ "Макинтош" подводит курсор с помощью "мыши" к соответствующей картинке на экране и нажимает встроенную в нее кнопку

менее утомительна и т. д. Устройство можно отсоединить от ЭВМ и вводить информацию с документов автономно. "Микрорайтер" имеет процессор, небольшую по объему память и экран на жидких кристаллах. Однако спрос на это устройство фактически вызван тем, что оно совершенно не похоже на обычную клавиатуру. Клавиатура – удел секретарей-машинисток, и руководитель не может снизойти до работы за клавиатурой, чего не учитывают специалисты по вычислительной технике.

Есть еще одно устройство, которое претендует на то, чтобы заменить клавиатуру. Его называют "мышь". "Мышь" представляет собой вариант потенциометрического пульта, который используется в компьютерных играх. Двигая рукоятку пульта, играющие перемещают по экрану курсор или какое-либо изображение, например изображение самолета. Однако вместо того, чтобы двигать рукоятку потенциометра, Вы можете перемещать само устройство по поверхности стола. При этом вращается шарик, укрепленный снизу устройства, и в ЭВМ поступает сигнал, управляющий перемещением курсора. От одной "мыши" пользы немного: должно быть

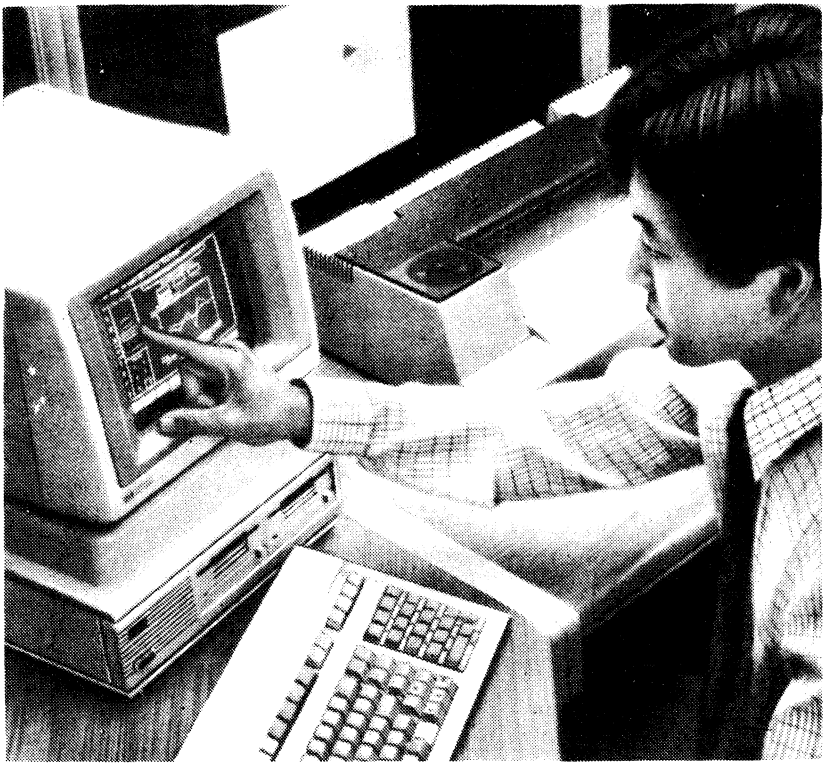


Рис. 11. Располагая необходимыми техническими средствами, пользователь может ввести данные в программу прикосновением пальца руки к экрану дисплея

еще программное обеспечение, создающее на экране образы, к которым следует подводить курсор. Например, в микроЭВМ "Лиза" и "Макинтош" фирмы "Эппл" операционная система "рисует" картины, с которыми работает "мышь", на экране. Вы видите на дисплее изображение корзины для бумаг, подводите к нему курсор, нажимаете кнопку, встроенную в "мышь", и ... выполняется операция по стиранию текущего файла. Результат и затраты времени на операцию те же самые, как если бы Вы набрали слово DELETE (стереть) на клавиатуре.

Еще один новый подход к вводу данных в ЭВМ связан с созданием сенсорного экрана. Невидимые инфракрасные лучи образуют своего рода решетку перед экраном. Когда Вы касаетесь экрана пальцем, Вы прерываете два пересекающихся в этой точке луча, и в программу автоматически вводятся координаты точки, которые сопоставляются с координатами блоков информации, изображенных на экране. В результате ЭВМ довольно точно отбирает необходимую информацию. Такое устройство может быть полезным при просмотре базы данных.

В качестве примера можно привести автоматизированную систему продажи авиабилетов, которую мне довелось недавно увидеть. Программное обеспечение системы обеспечивает четкую работу сенсорного экрана. Сначала на нем появляется обычное расписание рейсов с указанием времени, типа самолета, места назначения. Вы прикасаетесь пальцем к информации о рейсе, который Вам нужен. Датчики определяют место, где находится Ваш палец и сообщают об этом программе, которая сравнивает полученные координаты с координатами блоков информации о рейсах и выбирает интересующий Вас рейс. Экран очищается, затем на нем появляется план салона авиалайнера. Вы снова прикасаетесь к экрану, чтобы указать каким классом, в каком отделении (для курящих или некурящих) Вы хотите лететь и т. д. Экран снова очищается и перед Вами крупным планом часть салона самолета, где свободные места показаны голубым цветом, а проданные красным. Прикоснувшись еще раз к экрану, Вы окончательно выбираете место.

До этого момента все шло очень гладко и быстро, намного быстрее, чем при вводе данных с клавиатуры. Действительно, процедура, которой следовал пассажир, понятна четырехлетнему ребенку и основывается на простом принципе: видишь, что тебе нужно – дотронься до этого. Но в конечном счете Вы все равно должны набрать на клавиатуре свою фамилию, адрес, номер кредитной карточки.

Все больше проявляется увлечение ЭВМ, которые могут "говорить" и "понимать" речь. Теоретически, если Вы говорите с ЭВМ "нормально", они понимают и выполняют Ваши указания. Но на практике все получается иначе. Есть компьютеры, которые воспроизводят почти разборчивую речь, если вы введете английский текст. Однако, чтобы это произошло, программист должен ввести далеко не английские слова. Например, чтобы заставить ЭВМ членораздельно сказать "КОМПЬЮТЕР", Вы должны ввести "КА$_3$МПЬЮ$_1$ТЕР" (индексы означают различные оттенки одного и того же звука). Довольно трудоемкое занятие!

Электронная речь очень раздражает. Большим преимуществом надписей на экране является то, что их можно не читать, когда Вы знаете их содержание. Но если сообщение произносится ЭВМ, то каждый раз Вы вынуждены слушать скрежещущий монотонный голос: "Теперь, пожалуйста, введите номер Вашей кредитной карточки, будьте при этом внимательны". После пятидесяти раз у Вас появится желание взять в руки кувалду.

Но это еще не все. Очень трудно освободить разговорную речь от всякого рода междометий, придыханий, слов-паразитов вроде "хм", "ээ", "знаете", "поймите меня правильно". Невозможно без контекста различить на слух, скажем, слова "*plaice*" и "*place*". Только из контекста можно понять, идет ли речь о рыбе или о положении в пространстве. Пока ЭВМ не могут понимать речь (точнее, программисты не знают, как заставить ЭВМ это сделать).

Возможно я выгляжу скептиком и малосведущим человеком, но я не встречал еще ни одной ЭВМ, которая говорила бы сносно, или хотя бы не вызывала чувства раздражения.

Таким образом мы снова возвращаемся к клавиатуре. Несмотря на все попытки заменить клавиатуру, она просуществует еще много лет. Это отличное устройство для ввода информации. У него достаточно богатый набор знаков, чтобы ввести все тексты Шекспира, все адмиралтейские таблицы приливов адмиралтейства, все результаты футбольных матчей за год и избежать в то же время ненужных слов, вроде "хм", "ээ", о которых мы уже говорили.

1.6. МНОГОПУЛЬТОВЫЕ СИСТЕМЫ

Возможность хранить и находить информацию в базе данных дает каждому пользователю ЭВМ определенные преимущества. Эти преимущества возрастают, если базой данных пользуется несколько человек. При этом база данных выступает не только как "разумный" ящик для хранения картотеки, но и как средство мгновенного распределения информации. В качестве примера рассмотрим организацию системы информации в фирме одного знакомого литературного посредника. Он охраняет авторские права более 4000 авторов, начиная от известных в литературе имен и кончая мелкой сошкой, которая пишет книги о компьютерах. В составе фирмы восемь отделов, специализированных по странам и видам творческой деятельности (книги, пьесы, телепередачи, фильмы и т. д.). Авторское право может распространяться на несколько произведений, а число юридических лиц, добивающихся разрешения правообладателей на использование их произведений, может превышать несколько десятков. И все договоры связаны между собой. Допустим, телевидение ФРГ только что приобрело право на экранизацию нового романа Розы Бланделлы "Увядающие цветы". Возможно, это отразится на правах переводчиков в Бразилии, так как в этой стране показывается много телепередач из ФРГ. Сотрудники фирмы целый день бегают по этажам очаровательного четырехэтажного здания в стиле эпохи королей Георгов на берегу Темзы, чтобы взглянуть на картотеки друг друга и узнать о происшедших изменениях. Проблема может быть решена, если установить в здании несколько микроЭВМ, обрабатывающих информацию из единой базы данных. В этом случае, как только в базе данных будут сделаны дополнения к записи, касающейся мисс Бланделлы, они немедленно станут доступными для работы другим сотрудникам.

Многопультовые системы можно свести к следующим типам: многозадачные, многопроцессорные, локальные сети ЭВМ. В многозадачных системах несколько пользователей работают с одной ЭВМ в режиме разделения времени аналогично тому, как это делается на больших ЭВМ. В многопроцессорных системах несколько независимых ЭВМ объединяются в одном корпусе и имеют общую дисковую память, которая находится под управлением специальной ЭВМ. Как правило, в многозадачных и многопроцессорных системах пользователи работают с терминалами, находящи-

мися на некотором расстоянии от ЭВМ. В локальных сетях ЭВМ несколько автономных микрокомпьютеров соединяются между собой линиями связи.

Основная задача многопультовой системы – предоставить множеству пользователей возможность доступа к общим данным, хранящимся в базе данных. Многозадачные системы требуют очень мощных микроЭВМ, например построенных на базе микропроцессора 68000. На восьмиразрядных микроЭВМ многозадачная система, поддерживаемая операционной системой *MP/M*, приводила к длительному ожиданию абонентов, а обычные шестнадцатиразрядные процессоры не намного превосходят восьмиразрядные по быстродействию.

Стоимость микропроцессорной техники настолько низкая, что многопроцессорная система оказывается намного эффективнее. Каждый пользователь располагает своей ЭВМ и может пользоваться общим диском, считывая с него информацию одновременно с другими пользователями. Такой режим предъявляет определенные требования к программному обеспечению. Критический момент для системы управления базой данных наступает тогда, когда два пользователя одновременно хотят изменить или стереть одну и ту же запись. Программный аппарат базы данных должен быть в состоянии "блокировать" записи, чтобы не допускать тупиковых ситуаций для пользователей. Это означает, что если первый пользователь вносит изменение в запись, в ней появляется отметка, по которой программа закрывает доступ к этой записи для второго пользователя.

Практически система выдает программе ввода сообщение об ошибочной ситуации: "Запись блокирована". Что делать дальше? В каждой конкретной системе этот вопрос решается по-своему: неудачливый пользователь может подождать, а затем сделать новую попытку; он может прекратить решение задачи и заняться чем-либо другим. Для того чтобы система управления базой данных могла осуществить блокировку индивидуальных записей, она должна взаимодействовать с операционной системой, которая тем или иным способом блокирует файл в базе данных. Многопультовая система базы данных должна работать под управлением специальной операционной системы.

Еще одна, более скрытая конфликтная ситуация, возникающая в больших базах данных, которую мы упоминаем для полноты изложения материала, носит на профессиональном жаргоне название "мертвая хватка". Она имеет место тогда, когда два пользователя, обратившиеся к реляционной базе данных, пытаются изменить "виртуальную запись"*, состоящую из физических записей R1 и R2.

Программа пользователя *A* блокирует запись R1, а программа пользователя *B* блокирует запись R2. Оба затем пытаются блокировать оставшуюся запись и, конечно, терпят неудачу. Существует несколько реше-

---

* Слово "виртуальный" означает, что сущность, которую оно описывает, кажется неделимой, реальной, хотя на самом деле это не так. В данном случае две записи разделены, но пользователь может рассматривать их как единую запись.

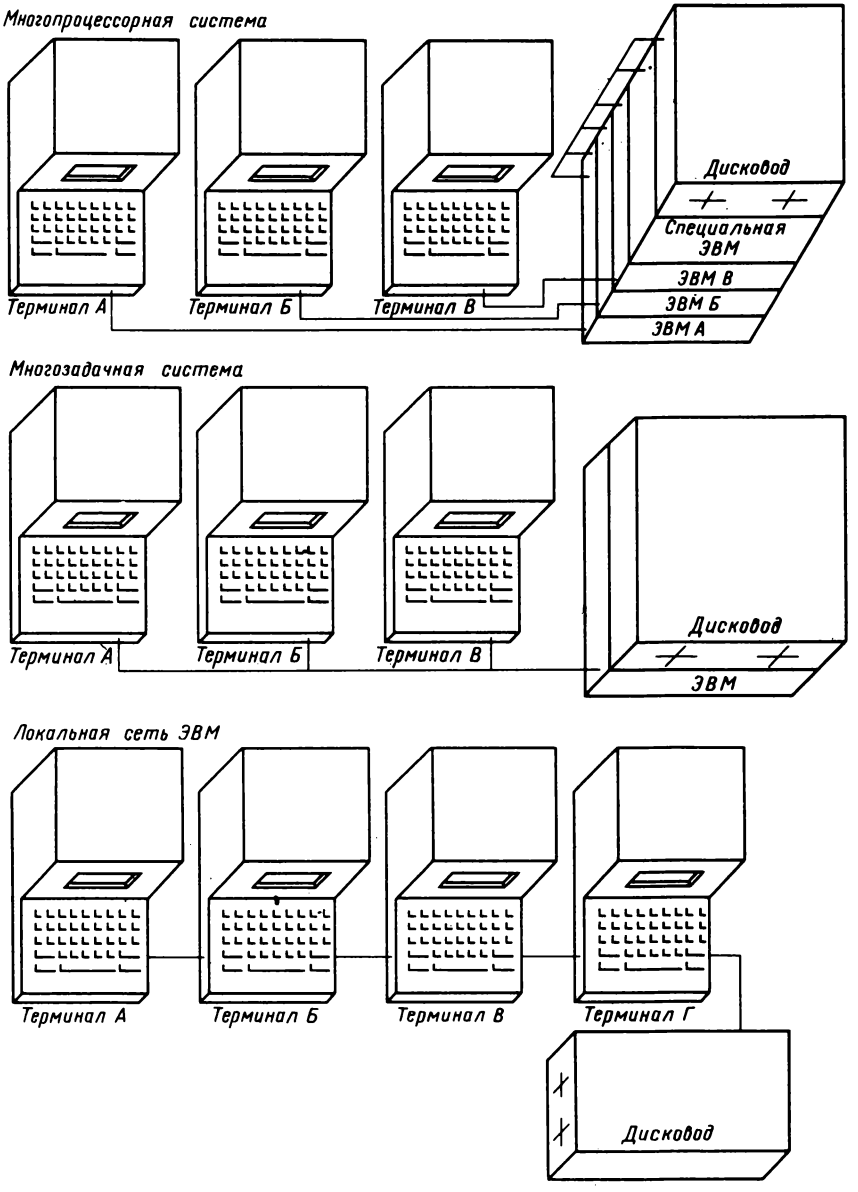


Рис. 12. Три основные схемы многотерминальных систем. При многопроцессорной обработке данных (а) каждый пользователь имеет собственную ЭВМ, которая получает данные от центральной ЭВМ. В многозадачной системе (б) каждому, пользователю по очереди выделяется процессорное время одной мощной микроЭВМ. В местных сетях (в) ЭВМ на рабочих местах связаны между собой и могут получать данные от центральной ЭВМ

ний проблемы, из которых самое простое заключается в том, что программа всегда блокирует запись, ближайшую к началу файла данных. В этом случае оба пользователя пытаются вначале блокировать первую запись. Тот кто не успевает этого сделать, оставляет другого в покое (вернее, это делает программа).

1.7. ВИРТУАЛЬНАЯ ПАМЯТЬ

Традиционные различия между тремя основными классами ЭВМ – большими, малыми и микро – быстро стираются. Однако, если для мощных миниЭВМ виртуальная память уже создана, то для микроЭВМ она только начинает создаваться.

Виртуальная память означает, что имеются технические средства, которые позволяют программисту использовать в машине столько памяти с произвольным доступом, сколько ему необходимо, не обращая внимания на физические размеры ОЗУ. Вы можете писать самые большие программы, и ЭВМ будет переносить их по частям с диска в оперативную память, и наоборот, создавая иллюзию, что память гораздо больше по размерам, чем она есть на самом деле. Поскольку пользователи решают на ЭВМ все более и более сложные задачи, размеры программ растут, и для небольших ЭВМ программистам приходится разбивать программы на сегменты. Это трудоемкая и сложная задача.

1.8. КАК ВЗАИМОДЕЙСТВУЮТ ЭВМ

Многопультовая система на основе микроЭВМ создает у пользователей впечатление, что каждый пользуется своей собственной ЭВМ и может более или менее свободно обмениваться информацией с другими машинами, расположенными в том же здании или на территории этого предприятия. Необходимость в обмене информацией электронным способом возникает и между удаленными друг от друга предприятиями и организациями.

Естественным каналом передачи данных являются телефонные линии связи. Действительно, телефонные провода тянутся от Вашего письменного стола к сотням миллионов других рабочих мест, во всех странах мира, и нет причин не использовать телефонные каналы общего пользования для передачи информации из ЭВМ.

Телефонная связь прочно и давно вошла в нашу повседневную жизнь, но она предназначена только для передачи человеческой речи. Когда Вы говорите в микрофон, то на его выходах появляется напряжение, которое меняется в соответствии с силой голоса. На другом конце канала связи, где-нибудь в Новой Зеландии, если Вы заказали трансконтинентальный разговор, напряжение меняется аналогичным образом. Уже на гораздо меньших расстояниях возникает множество искажений в виде свиста, щелчков, эха. Телефонная сеть имеет ограниченную полосу пропускания звуковых частот, поэтому голос, который Вы слышите в телефонной труб-

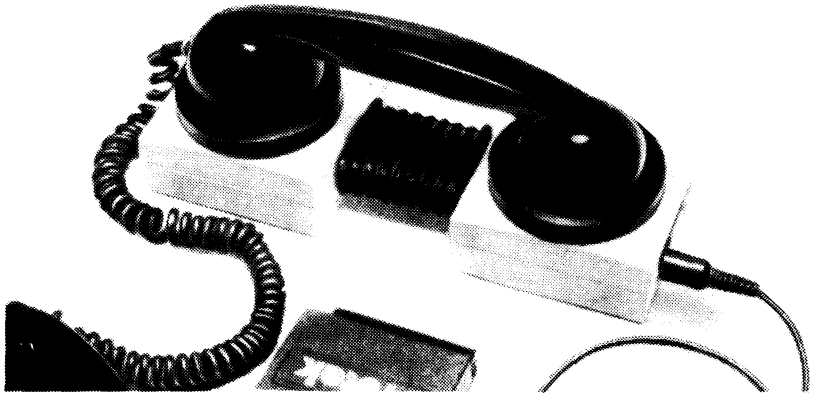


Рис. 13. Модем преобразует импульсные сигналы в звуки низкой частоты, которые передаются по телефонным каналам связи. Модем позволяет удаленной ЭВМ обмениваться данными с центральной базой данных

ке, имеет мало общего с настоящим голосом собеседника, хотя его можно узнать. Телефонная связь обеспечивает передачу человеческой речи, но она не приспособлена для передачи данных из ЭВМ.

Напомним, что компьютеры различают только импульсы постоянного тока, означающие двоичные коды 0 или 1, и не реагируют на силу человеческого голоса. Выход заключается в том, чтобы использовать специальное устройство, называемое "модемом" (модулятор-демодулятор), которое преобразует два сигнала ЭВМ в два музыкальных звука. Модем "насвистывает" в телефон, а второй модем, расположенный на другом конце провода, переводит эти звуки в нули и единицы.

Из-за больших искажений в телефонных линиях нельзя надеяться, что информация будет передана без ошибок. Модемы и их программное обеспечение постоянно проверяют, не произошли ли сбои в передаче информации. Основным методом проверки является метод "контрольных сумм".

Все единицы в двоичных кодах складываются и сумма передается вместе с сообщением. Если на другом конце линии сообщение не искажено, новая контрольная сумма должна совпасть с переданной суммой. Если суммы не равны, то передающая машина получает сообщение "не понимаю" и должна повторить передачу информации. Все это происходит автоматически. Пользователь ничего не видит, но зато удивляется, почему на все уходит так много времени. Телефонные каналы имеют еще один большой недостаток – скорость передачи данных по ним очень мала. Максимальная скорость, с которой Вы можете надежно пересылать данные, составляет 300 бод (примерно 300 бит/с). Это означает 30 байт/с или 5 слов текста. Чтобы передать 10 000 слов, потребуется более получаса.

Новейшие технические средства обеспечивают большие скорости передачи данных. В Англии телефонная система включает коммутируемую сеть пакетной передачи данных, в которой информация передается со скоростью 50 000 бит/с. Абоненты сети расположены по всей стране и за рубежом. Вы можете выйти в сеть через местную АТС, набрав соответствующий номер телефона. Однако заметного выигрыша в скорости передачи данных не получается, так как сказываются ограничения, накладываемые местной АТС. Если Вы пересылаете большие массивы данных, то за несколько тысяч фунтов стерлингов Вы можете получить доступ к сети по специальному каналу связи. Однако большинству пользователей это не по карману.

Одна из причин, по которой крупные телефонные компании занимаются сейчас прокладкой оптоволоконного кабеля, заключается в том, что по нему можно передавать данные с большой скоростью по некоммутируемым каналам для всех пользователей. Оптоволоконный кабель толщиной с человеческий волос может передавать данные в виде световых импульсов со скоростью несколько миллионов бит в секунду. Одновременно по кабелю могут передаваться данные для ЭВМ, телевизионные изображения и телефонные разговоры – все в цифровой форме.

Современные модемы встраиваются в ЭВМ и внешне напоминают гнездо для телефонной трубки. Вы набираете номер, по которому хотите передать данные, включаете в сеть ЭВМ и ... уходите. Есть модемы, которые могут набрать телефонный номер самостоятельно. Они также различают гудки по телефону и могут сообщить, что абонент занят, линия неисправна и т. п. Для работы модемов необходимо программное обеспечение, позволяющее подсчитывать контрольные суммы, набирать номер и в ряде случаев расшифровывать получаемые из системы сигналы.

1.9. СОВРЕМЕННЫЕ УСТРОЙСТВА ВНЕШНЕЙ ПАМЯТИ ЭВМ

При создании баз данных Вы заинтересованы в хранении большого объема информации. Технически не существует ограничений на количество пакетов дисков, которые Вы можете использовать в базе данных. Их число ограничивается их стоимостью. Говорят, что база данных Министерства внутренних дел Великобритании располагает достаточным числом дисков, чтобы хранить записи в 200 ÷ 300 слов текста по каждому из 55 миллионов жителей Британских островов. Это составляет около 80Г байт (тысяч миллионов байт). Нет никакого сомнения, что в США подобная база данных хранит еще больший объем информации.

Хранение информации требует затрат, и чем больше Вы можете затратить на это денег, тем более совершенную технологию Вы можете использовать. Существует определенная иерархия стоимости хранения информации:

1) каталоги в библиотеках;
2) документационное обеспечение аппарата управления;

3) микрофиши;
4) дисковая память ЭВМ;
5) оперативная память ЭВМ с произвольной выборкой.

Бумага в обозримом будущем остается самым дешевым средством долговременного хранения информации в архивах. Она может оказаться полезнее электроники и во многих других случаях. Вряд ли кто из нас, желая почитать перед сном, возьмет в постель дисплей, а не книгу. Газета также удобный источник информации, который очень трудно заменить. Ее преимуществами кроме низкой цены являются: компактность, легкость, с которой Вы просматриваете базу данных, содержащуюся в газете, например раздел "Куда пойти отдохнуть".

Предположим, что повсюду используются только электронные средства информации, и вдруг кто-то вновь изобретает бумагу. Подлинный переворот в информатике! Стоимость хранения – ничтожные доли пенса за слово! Устройства ввода-вывода вообще не требуются! Вы пишете с помощью обугленной палочки! Вы читаете невооруженным глазом! Сроки хранения исчисляются тысячелетиями! Величайшее открытие на все времена!

Но вернемся из области фантазии к реальной технологии. Хранение информации на микрофишах – это нечто среднее между бумажной и электронной технологией. Микрофиша представляет собой миниатюрную фотографию документа. В настоящее время существуют системы, в которых индексы хранятся на диске, а сама информация – на микрофишах. Система управления базой данных и специальные технические средства обеспечивают автоматический поиск необходимой микрофиши.

Стоимость оперативной памяти в расчете на одно слово хранимого текста составляет 3 пенса* и постоянно снижается. Но пока ОЗУ не могут быть использованы для длительного хранения данных, скажем, в течение нескольких лет. Кроме того, любое расширение основной памяти ЭВМ используется программистами для улучшения методов обработки данных.

Разработчиков баз данных интересуют прежде всего дисковые устройства памяти. Дисковая память развивается по нескольким направлениям и бесполезно предугадывать, какое из них в конечном итоге возьмет верх. Безусловно, общим является стремление разработчиков разместить больше информации на диске обычных размеров. Важно понять, что увеличение размеров диска нежелательно, поскольку эти размеры определяют габариты всего блока. В электронной промышленности сложилась довольно устойчивая цена за 1 кг изделия, поэтому увеличение габаритов означает увеличение стоимости изделия. Кроме того, чем больше размеры блока, тем больший путь проходит магнитная головка, тем длительнее время поиска. В производстве дисков для ЭВМ стремятся уменьшить цену не за счет снижения себестоимости изделий, а за счет размещения на них большего количества информации. Этого можно добиться, изменив плот-

*1 пенс = 1/100 фунта стерлингов. – *Прим. пер.*

ность записи. Чтобы лучше понять возникающие при этом проблемы, рассмотрим, каким образом можно поместить на обычный диск больше информации.

Магнитное вещество, нанесенное на поверхность диска, намагничивается в определенной зоне. Эту зону можно расширить, если магнитную головку переместить ближе к краю или центру диска. Однако линейные скорости вращения диска относительно магнитной головки будут значительно отличаться по мере ее движения от края к центру диска. Каждый бит информации пишется за фиксированный промежуток времени. И длина намагниченного участка, соответствующего одному биту, намного больше на периферии, чем в центре диска. По мере того как головка смещается к центру, длина участка уменьшается и доходит до пределов, когда его магнитное поле уже не улавливается. Чем ближе к центру находится головка, тем больше требуется времени для записи и чтения данных. Это означает, что запись на наружных дорожках не такая плотная, как могла бы быть.

Одно из возможных решений — заставить диск вращаться с разной скоростью, в зависимости от положения магнитной головки. Такая конструкция реализована фирмой "Сириус", и позволяет разместить на диске гораздо больше данных, но при этом время доступа возрастает, так как накопитель должен изменить частоту вращения вала при перемещении головки. Частоту вращения нельзя изменить мгновенно, необходимо ждать, пока она стабилизируется, и только после этого перемещать головку для записи или чтения данных.

Другой способ увеличить емкость накопителя на магнитных дисках — сделать головку тоньше, чтобы она писала более узкие дорожки. Тогда число дорожек на диске может быть увеличено. Добавьте к этому хитроумные методы сжатия информации и Вы получите диски с "удвоенной" и "учетверенной" плотностью записи. Все это хорошо в теории, но когда дорожка становится тоньше, головку необходимо позиционировать очень точно, чтобы считывать данные. Первым симптомом неправильной настройки головки являются сбои при чтении диска, записанного на другой машине — и все из-за того, что положение головки на несколько тысячных долей сантиметра отличается от эталонного. Даже в тех случаях, когда различные машины настроены одинаково, инженеры-электронщики вынуждены преодолевать большие трудности. Магнитные головки должны находиться в том же самом положении на диске, даже когда температура внутри блока меняется от -20 до $+40\ ^{\circ}C$. Такой перепад температур возможен зимой, если пользователь оставит портативную ЭВМ в салоне автомобиля, а утром приедет на работу и в помещении с центральным отоплением включит ЭВМ. Она или он будут очень удивлены и раздражены, если машина не будет работать так, как описывается в руководстве. Можно уменьшить зазор между диском и головкой, при этом участки намагничивания становятся меньше и их общее число на диске возрастает. Для гибких дисков этот способ мало подходит, так как поверхность диска испытывает сильное биение. Нужна твердая гладкая поверхность, чтобы мак-

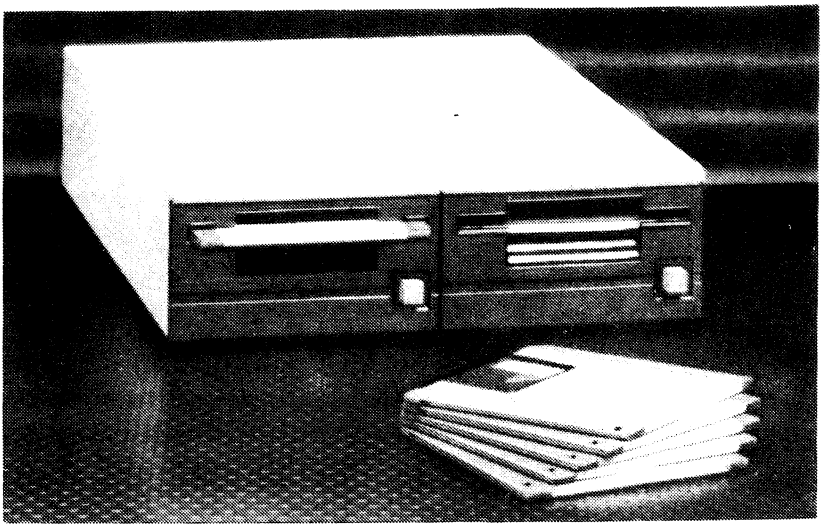


Рис. 14. Диски становятся меньше по размерам, а плотность записи на них повышается. Многие новые микроЭВМ снабжены дисководами для дисков 82,5 мм

симально приблизить к ней головку. Дисковод должен быть герметично закрыт, чтобы исключить попадание частиц пыли в зазор между диском и головкой.

Фирма "Винчестер" создала диск, емкость которого увеличена, но ей пришлось отказаться от гибкой конструкции. Диски, вмещающие информацию 5—20М байт широко распространены. Если набрать на одном валу пакет дисков, то емкость накопителя с обычными дисками диаметром 5 1/4 дюйма (133 мм) может составить 300М байт.

До настоящего времени промышленность выпускает магнитные диски, на которых намагниченные участки располагаются параллельно поверхности диска.

Если бы магнитные участки располагались в теле диска (этого можно достигнуть, поместив северный полюс электромагнита над поверхностью диска, а южный под ней), то потребовалась бы значительно меньшая площадь намагничивающей зоны при одном и том же объеме намагниченного материала. Емкость одного диска диаметром 133 мм при этом можно довести до 30М байт, а емкость накопителя до 1200М байт.

Ведется также поиск в направлении создания оптических дисков. Лазерная технология сейчас используется для записи музыки и ее начинают энергично применять для записи данных. Во многих отношениях эта технология выглядит идеальной. Информация записывается путем прожигания мощным лазерным лучом небольших углублений на поверхности диска.

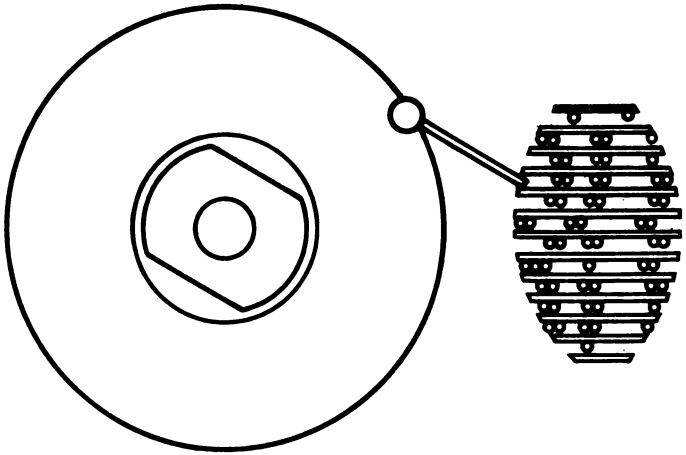


Рис. 15. Разрабатываемые в настоящее время лазерные диски обеспечивают гораздо большие объемы памяти, чем обычные диски. Луч лазера записывает двоичные цифры в виде крошечных углублений на поверхности диска

Считываются биты лазером небольшой мощности. Ничто не касается поверхности диска, не вызывает его износа. Размеры углублений очень малы, поэтому точность записи на диске может быть большой. Для высококачественной записи музыки в цифровой форме необходимо в течение 1 с записывать на диск около 50000 бит, поэтому небольшой диск с продолжительностью звучания 1 ч будет вмещать 22М байт данных. Диски больших размеров будут иметь емкость около 1000М байт.

На пути новой технологии возникают очень сложные проблемы. Пока информация на оптические диски может быть записана только в производственных условиях. Музыку записывать на заводе можно, а вот для пользователей ЭВМ такой способ записи их информации не подходит. Ведутся интенсивные разработки по совершенствованию технологии записи и уже есть некоторые успехи. Однако на выпускаемых в настоящее время промышленностью дисках запись можно сделать только один раз. В отличие от магнитных дисков в оптических дисках нельзя использовать поверхность диска повторно. Но это не так плохо. Исчезает проблема, когда на диске уничтожаются файлы, которые впоследствии могут понадобиться. Потребуется внести незначительные изменения в операционную систему, чтобы учитывать номер версии файла. Например, Вы можете указать операционной системе выдать восьмую, самую последнюю версию интересующего Вас файла.

Другой недостаток заключается в том, что процесс записи ведется довольно медленно. Если для среднего по размерам диска "Винчестер" требуется 20 мс, чтобы записать на диск блок информации, то для оптического диска потребуется 100–200 мс. Такая скорость записи может быть приемлемой для основного файла базы данных. В конце концов, 200 мс, или 0,2 с намного меньше времени, необходимого для печати фактической записи на машинке и расстановки карточек в картотеке. Но, как мы уже знаем, СУБД создает множество служебных файлов, в том числе индексных. При такой скорости записи процесс добавления к базе данных новой записи потребует нескольких минут, так как надо скорректировать все индексы. Опыт показывает, что даже 5 с кажутся вечностью, когда Вы сидите перед дисплеем. Одно из решений заключается в том, чтобы хранить индексы на диске типа "Винчестер", а информацию – на оптическом диске. Но если в базе данных информация индексируется полностью, то сам объем индексов может стать больше, чем объем данных, и возникает новая проблема – где разместить индексы. Можно попытаться вернуться к старому понятию "ключевых полей" и индексировать только значения этих полей, но тогда теряется преимущество гибкого поиска, которое дает современная СУБД.

1.10. ОШИБКИ В ПРОГРАММАХ

Любая программа содержит ошибки. Проявляются они в том, что программа работает не так, как было задумано программистом. Результаты ошибок в программах непредсказуемы. В тяжелом случае единственная незаметная маленькая ошибка в коде может уничтожить все файлы на Вашем диске. В лучшем случае ошибка может иметь столь несущественные последствия, что ее можно игнорировать.

В некотором смысле программа напоминает карточный домик: тот факт, что домик стоит, еще не гарантирует, что он не рассыплется в следующее мгновение. Программы опровергают опыт нашей жизни. Обычно, если что-то работает, то оно работает. Если новый стул выдержал Вас, он выдержит Вас и в следующий раз: если сошедший с конвейера автомобиль проехал один километр, он сможет проехать еще сотни километров; если здание простояло 5 мин, то, как уверяют строители и архитекторы, оно простоит еще сто лет. Однако на основании того факта, что часть программы работает, ничего нельзя сказать о работоспособности остальной части программы. СУБД может насчитывать десять тысяч команд и в каждой из них может быть любое количество ошибок. Каждую ошибку надо найти и исправить вручную. Процветающие фирмы, занятые разработкой программного обеспечения, специально нанимают профессионально неподготовленных людей, чтобы они поработали с вновь созданными программами. В их задачу входит за короткое время сделать столько неправиль-

ных обращений к программе, сколько пользователь не сделает за долгий период. Когда программа запрашивает цену товара, оператор набирает на клавиатуре слово "Почему" вместо числа; когда надо выбрать в меню функцию, помеченную определенной буквой,он нажимает клавиши *УПР, Z* и смотрит, как на это реагирует программа.

Ошибка в программе не всегда бывает по вине программиста. Работа типичной прикладной программы для микроЭВМ зависит от работы следующих программ:

1) операционной системы;

2) программного интерфейса между операционной и машинной системами (обычно называемый *BIOS*);

3) транслятора с языка, на котором написана программа;

4) модулей, написанных другими программистами и вставленных в программу.

Все эти программы были написаны разными людьми и в разное время и все имеют свои собственные ошибки. Мало того, могут быть скрытые ошибки, которые проявляются только тогда, когда Ваша программа взаимодействует с новыми версиями этих программ. Например, программа может идеально работать на машине разработчика, но когда Вы загружаете ее в Вашу ЭВМ с той же операционной системой, программа ведет себя странно, потому что сказалась скрытая до сих пор ошибка в операционной системе. Переводить задачи с одной машины на другую – это очень трудная дорогостоящая работа, требующая высокой квалификации. Не верьте тому, кто утверждает, что это легко сделать.

От ошибок могут избавить только время и опыт. По этой причине новые, пусть даже замечательные программы нежелательны. Они должны, они "обязаны" быть с ошибками. Гарантией надежности программ являются длительное время их применения и опыт многих пользователей.

После того как ошибки найдены, разработчики программ ведут себя по-разному. Некоторые фирмы закрывают глаза на их существование, другие ограничиваются выпуском извещений об ошибках и вносят исправления раз в год или реже в новые версии программ, третьи начинают их немедленно устранять. Все эти действия носят название "программное сопровождение". Прежде чем купить пакет программ управления базой данных и доверить ЭВМ дорогую для Вас информацию (прикиньте, сколько будет стоить 50 000 записей в базе данных, если стоимость приобретения и ввода одной записи составляет 20 пенсов), Вы должны точно узнать, какое сопровождение Вам будет обеспечено.

Справедливости ради отметим, что у фирм, занятых разработкой программ, имеются свои проблемы. Прежде такая фирма продавала пользователям больших и средних ЭВМ 20 или 30 копий программы и получала за каждую копию большие деньги – до миллиона фунтов стерлингов. Пользователь платил солидную сумму ежегодно по договору на обслуживание, и за счет этих средств фирма могла посылать программистов в ВЦ пользователя исправлять ошибки по мере их обнаружения. Сегодня, ког-

да тысячи людей покупают программы по цене несколько сотен фунтов, а скидка в торговой сети сводит эту сумму до десятков фунтов за копию невозможно обслуживать каждого клиента индивидуально. Отдельным актом этой большой драмы, называемой "Программное сопровождение", выступает "поддержка" конечного пользователя. Поддержка необходима, когда Вы не можете разобраться с программной документацией или разобрались, но программа не работает. В идеальном случае Вы хотели бы позвонить специалисту, который знает программу и может помочь Вам устранить любые трудности, связанные с ее внедрением. Фирма, создавшая пакет программ, наоборот, хотела бы избежать такого рода услуг, так как подобные специалисты редки и дорого стоят. Можно ожидать такой поддержки от того, кто продал Вам пакет. Но чаще всего это мелкий торговый посредник, не знающий всех пакетов программ, которые он продает. (Программисту-профессионалу требуется не менее месяца напряженного труда, чтобы понять, как работает пакет программ.) Прежде чем купить пакет программ управления базой данных, выясните, кто подскажет Вам, как заставить его работать, если Вы сами этого не сможете сделать.

1.11. ЗАТРАТЫ НА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

В клубок проблем, связанных с программным сопровождением и поддержкой, вплетается и рост затрат на программное обеспечение. Все знают, что цены на средства вычислительной техники постоянно снижаются. Можно было бы ожидать подобного снижения цен и для программных средств. В конце концов, программный продукт не имеет материального выражения, если не считать носителя, на котором записана программа. Например, пакет программ управления базой данных может стоить 500 фунтов. Диск, на котором хранится система, стоит около 1 фунта. Трудозатраты на копирование программ составляют не более 5 мин (или примерно 4 фунта), а документация, которая поставляется вместе с диском, стоит 5 фунтов. За что пользователь платит остальные 490 фунтов? Существует два объяснения такой, с первого взгляда, высокой рентабельности. Во-первых, программное обеспечение становится все более и более сложным. Разработка пакета и его сбыт требуют больших затрат. Надо затратить примерно 1 млн. фунтов на разработку современного пакета программ для решения экономических задач и еще два-три млн. на создание рынка сбыта. Во-вторых, раньше, если Вы покупали ЭВМ и пакет программ у солидной фирмы, Вы еще ежегодно платили ей за сопровождение Вашего программного обеспечения. (Сейчас это также практикуется для серьезных пакетов программ для микроЭВМ, например трансляторов). Эти взносы давали возможность фирме-поставщику содержать программистов, которые были готовы отвечать на Ваши наивные вопросы по телефону или могли подъехать к Вам и отладить программу. Теперь, когда число клиентов переваливает за тысячи, фирмы не занимаются сбором подобных взносов. Стоимость сопровождения входит в начальную цену пакета. Более того, пользователи стремятся заплатить более высокую цену,

лишь бы не терять уверенность в том, что в нужный момент представители фирмы будут рядом. Оценивая затраты на приобретение пакета программ, необходимо учитывать скрытые расходы на обучение работе с пакетом. Лишь немногие пользователи-программисты могут освоить пакет программ для решения экономических задач менее чем за 3 мес. Зарплата персоналу за 3 мес. и потерянная прибыль составляют примерно 5000 фунтов. К этому надо добавить еще стоимость и ценность данных, введенных в систему. Затраты на сбор и подготовку данных могут значительно превышать остальные виды затрат.

1.12. СБОИ И ОТКАЗЫ ЭВМ

Теоретически данные на диске могут храниться бесконечно долго. На практике это далеко не так. Хотя намагниченные участки обладают способностью к самоподмагничиванию, крошечные поверхности, которые они занимают, постепенно теряют магнитные свойства. Вот почему архивы на магнитных носителях переписываются каждые два года.

Еще большей проблемой являются сбои и отказы ЭВМ. Говорят, что система "отказала", когда она перестает работать. Такое неприятное положение может быть вызвано многими причинами. В простейшем случае кто-то отключил питание во время расчетов. В большинстве случаев ошибка в программе приводит к ее аварийному завершению. Самое неприятное происходит тогда, когда головка чтения–записи "продирает" поверхность диска. Но какая бы причина ни была, результатом скорее всего будет уничтожение файлов на диске. Если Вы доверили ЭВМ серьезные задачи, такого рода сбой воспринимается, как удар по затылку. Так же болезненно воспринимается и то, что под угрозой оказывается метод работы, который стал частью Вашего "я".

Если бы мы относились к жизни философски, то могли бы ожидать эти бедствия, поскольку ЭВМ необычайно сложные машины. Если собрать микроЭВМ из механических деталей, вроде шестеренок и зубчатых колес в часовом механизме, она была бы размером с концертный зал. Удивительно вообще, что микроЭВМ работает, не говоря уже о том, что она работает надежно и неопределенно долго. В реальных условиях все компьютеры ломаются время от времени. Когда это случается, файлы могут быть повреждены или вообще уничтожены. Нетрудно представить себе, как легко ошибка в операционной системе может перемешать все в файлах, имеющих сложную организацию. Искушенные пользователи понимают, что их файлы могут исчезнуть в любой момент. Они копируют их каждый день с тем, чтобы в худшем случае потерять только результаты работы одного дня. Если их операционная система (например, *Unix*) отмечает каждый файл, в который были внесены изменения в течение дня, то в конце дня оператор может автоматически копировать все файлы с изменениями.

Предусмотрительные пользователи раз в неделю копируют все свои файлы. Они хранят предпоследние копии дома, а последние копии – на

работе. И если на работе случится пожар, надо будет только восстановить информацию за последнюю неделю. Такая система поддержания файлов известна под названием "дед, отец, сын". Надо исходить из того, что ЭВМ может полностью испортить базу данных в любую минуту.

Процесс поддержания базы данных включает копирование содержимого дисков (обычно твердых) на гибкие магнитные диски. Большинство пользователей микроЭВМ используют специальные устройства, которые выполняют эту монотонную работу. Полезно приобрести специальное устройство перезаписи информации с диска на ленту, которое выполнит эту операцию за несколько минут.

ГЛАВА 2
СИСТЕМА УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ

По существу, база данных – это совокупность записей, хранящихся в ЭВМ. Записи должны быть похожи по форме, но безусловно должны различаться по содержанию. В базе может быть несколько типов записей (см. гл. 4).

Управляет работой базы данных специальная программа, которая организует и хранит данные таким образом, чтобы можно было ввести данные с клавиатуры или другого устройства ввода информации, снова найти и извлечь их при необходимости. Программа оказывает нам такую же помощь, как и добросовестный конторский служащий. В состав пакета программ для управления базой данных микроЭВМ обычно входят следующие программы:

1) ядро системы, которое обеспечивает хранение и поиск информации в базе данных. Эта программа часто использует сообщения "языка запросов" (см. гл. 5), с помощью которого пользователь обращается к базе данных;

2) программа отображения на экране формы для ввода и поиска данных;

3) генератор отчетов, с помощью которого пользователь печатает "отчеты", т. е. перечни или таблицы с записями, удовлетворяющими заданным условиям поиска;

4) средства взаимодействия ядра пакета и программ обработки текстов, позволяющие СУБД не только выдавать списки, скажем адресов и фамилий, но и печатать текст, например деловые письма. Такое взаимодействие станет возможным уже в ближайшее время (см. гл. 3).

Однако единой стандартной СУБД не существует и состав пакетов программ, написанных различными авторами, отличается от предложенной схемы.

2.1. "РАЗУМНЫЙ" ЯЩИК ДЛЯ ХРАНЕНИЯ КАРТОТЕКИ

Функции программы, управляющей базой данных, можно, в какой-то мере, сравнить с функциями картотечного ящика, наделенного способностью проверять информацию при заполнении картотеки и осуществлять просмотр ее содержимого. Например, Вы можете хранить в базе данных списки фамилий и адресов клиентов фирмы, избирателей в округе. Или это может быть каталог книг в библиотеке, список компаний и их директоров и т. д. Главное отличие компьютеризованной базы данных от карточных или печатных каталогов и указателей заключается в том, что Вы можете задавать самые разнообразные вопросы. "Найдите книги всех авторов, чьи фамилии начинаются с буквы Д, заканчиваются на букву Е и изданы в Америке в 1952 г." или "Найдите все предприятия нефтяной промышленности, которые закупили продукцию нашей фирмы больше чем на 10 000 фунтов стерлингов и имеют конторы там-то и там-то" – такого рода запросы легко выполнить в электронной базе данных и совсем непросто в бумажной информационной системе.

Ядро пакета программ управления базой данных настолько сложное, что наверняка Вы приобретете эту программу у фирмы, специализирующейся на разработке программного обеспечения. Получив программные средства, конечный пользователь определяет их конфигурацию, после чего он должен сообщить системе, что в ней будет храниться информация, скажем, о книгах или о заказчиках.

Именно определение необходимой конфигурации программных средств делает пакет управления базой данных более интересным и более сложным приложением, чем применение программ обработки текстов или программ расчета рабочих таблиц, которые сегодня пользуются наибольшим спросом у владельцев микроЭВМ. Эти программы выполняют

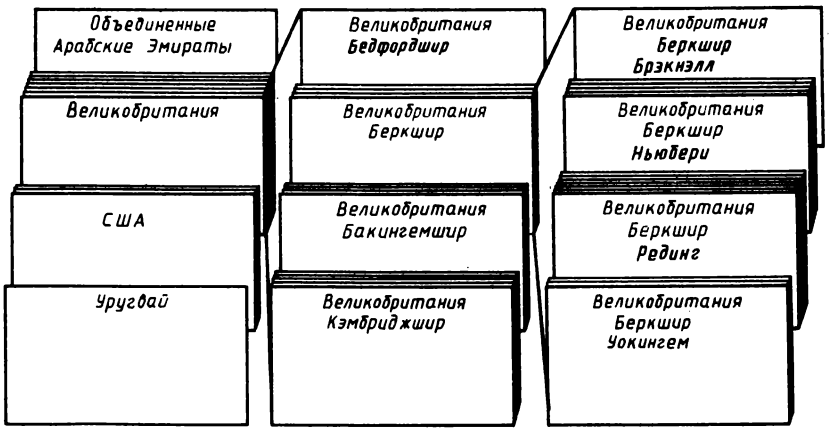


Рис. 16. Базу данных можно уподобить картотеке, организованной таким образом, чтобы можно было найти нужную запись по одному из ее элементов

определенные функции независимо от специфики задач пользователя, поэтому в них легче разобраться. Работа базы данных во многом зависит от конфигурации программных средств, установленной пользователем, что требует более глубоких знаний и большей ответственности.

Необходимо учитывать, что стоимость информации может быть значительной. Потеря информации, содержащейся в деловом письме или разработочных таблицах, подготовленных с помощью микроЭВМ, не выглядит катастрофой, поскольку в течение часа или двух они могут быть подготовлены заново. С другой стороны, в базе данных для современной микроЭВМ может храниться до полумиллиона записей. Если затраты на ввод каждой записи составляют 10 пенсов, то полная база данных будет стоить 50 000 фунтов стерлингов – гораздо больше стоимости технических средств и, как это ни прискорбно для фирм, разрабатывающих программное обеспечение, намного превысит стоимость программ. Вполне возможно, что в базе данных хранится информация, потеря которой невосполнима. Поэтому, если Вы собираетесь создавать базу данных, то стоит уделить ей некоторое внимание.

Однако хватит предупреждений. Чтобы создать небольшую базу данных, достаточно задать на экране "форму", в соответствии с которой будет вводиться и отображаться информация, и "отчет", который будет печатать микроЭВМ (подробнее см. гл. 3). Обычно представление пользователя о работе базы данных этим и ограничивается, однако мы рассмотрим, что происходит внутри микроЭВМ. Сердцевиной базы данных является основной файл, в котором хранится вся информация. От организации этого файла зависит очень многое. Для примера рассмотрим базу данных, состоящую из фамилий и адресов. Подобные базы, возможно, самые распространенные, хотя и необязательно самые простые.

При традиционном подходе к организации базы данных прежде всего необходимо решить, какие элементы информации Вы собираетесь хранить и какой они будут длины. В результате получится база данных самой простой организации – с записями фиксированной длины. Пусть в ней хранится следующая информация:

Элемент записи	Длина элемента	Длина записи нарастающим итогом
Первое имя	10	10
Инициал	1	11
Фамилия	15	26
Адрес 1	15	41
Адрес 2	15	56
Адрес 3	15	71
Штат	15	86
Почтовый индекс	7	93

Основной файл будет выглядеть следующим образом.

Конец 13-й записи 14-я запись

85	86	87	88	89	90	91	92	93	Д 1	Ж 2	О 3	Н 4	5	6	7	8	9	10	П 11	Д 12	А 13	В 14	Е 15	Н 16	П 17	О 18	Р 19	Т 20	21	22	23	24	25	26
									Имя										Инициал				Фамилия													

1 27	6 28	29	Л 30	И 31	Н 32	Д 33	Е 34	Н 35	36	Д 37	Р 38	39	40	41	О 42	У 43	К 44	С 45	А 46	Й 47	Д 48	49	50	51	52	53	54	55	56
Адрес 1															Адрес 2														

Конец 14-й записи

П 57	А 58	С 59	А 60	Д 61	Е 62	Н 63	А 64	65	66	67	68	69	70	71	К 72	А 73	Л 74	И 75	Ф 76	О 77	Р 78	Н 79	И 80	Я 81	82	83	84	85	86	С 87	А 88	1 89	2 90	3 91	4 92	5 93
Адрес 3															Штат															Почтовый индекс						

С 1296	А 1297	1 1298	2 1299	3 1300	4 1301	5 1302

13-я запись (от начала файла)
× 93 знака + 87 знаков =

Пробелы обозначают незаполненные места в каждой зоне или "поле" записи. Обратите внимание на то, что за инициалом "П" вплотную следует слово "ДАВЕНПОРТ". Программа знает, что под инициал отведен один символ и сможет его выделить. В поле, содержащем почтовый индекс, нет незаполненных мест, поскольку в США действует семизначный индекс (записи в базе данных Министерства связи США также фиксированной длины).

Важно знать длину каждого элемента хранящейся информации, так как в этом случае ЭВМ легко может найти любые данные. Допустим, требуется найти в файле почтовый индекс 14-го адресата. В каждой записи почтовый индекс начинается с 87-го, заканчивается 93-м символом. Поскольку программе известны структура и длина каждой записи, ей остается только пропустить $13 \times 93 + 87 = 1296$ знаков в файле и следующие семь символов образуют искомый почтовый индекс.

Во многих ранее разработанных базах данных значения одноименных полей записей собираются в отдельные файлы. Например, все фамилии хранятся на диске в одном месте. Если Вы запрашиваете 14-ю запись, то система собирает необходимую информацию, прочитывая 14-е поле в каждом из соответствующих файлов. Такая организация массивов информации в базе данных носит название "параллельной" (она будет рассмотрена в гл. 4).

В реальной жизни Вас не интересует номер записи. Вас интересуют данные, связанные, скажем, с фамилией Давенпорт, и Вам совершенно безразлично, где эта запись расположена в базе данных. Нужна какая-то схема индексирования. В простейшем случае Вы можете решить, что всегда будете осуществлять поиск интересующей Вас записи по фамилии, как это делается в алфавитном каталоге. После того как система запросит у Вас фамилию, а Вы наберете на клавиатуре слово "Давенпорт", программа просмотрит во всех записях поля, содержащие фамилии (символы с 12-го по 26-й) и определит интересующую Вас запись.

Поле, содержащее фамилию, называется ключевым полем, или ключом. По ключу мы отыскиваем необходимые данные. Только самые устаревшие программы управления базами данных осуществляют поиск по единственному ключу. Весь смысл применения ЭВМ для хранения и поиска информации заключается в том, чтобы находить информацию по другим признакам, отличным от фамилии или номера документа. Иначе было бы достаточно печатного указателя. Некоторые пакеты управления базами данных, например *dBaseII* допускают до семи ключей в записи, другие, скажем *Superfile*, позволяют использовать любое число ключевых полей (см. § 3.1). Такой метод доступа к записям базы данных, когда ЭВМ просматривает все записи подряд, называется "последовательным". Это самый простой и понятный способ, но он требует чрезвычайно много времени, если хранятся большие объемы информации.

Более производительный способ заключается в создании отдельного индексного файла, который содержит значение всех ключевых полей или их сокращение вместе с адресами, указывающими начало каждой записи.

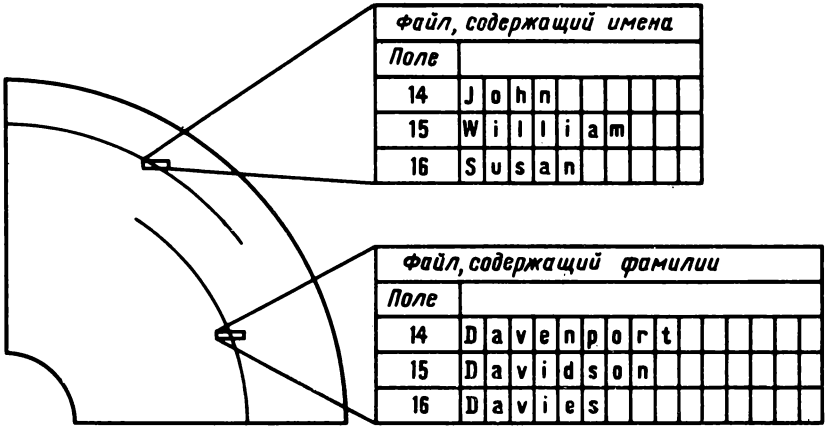


Рис. 17. В традиционных базах данных значения различных полей записи хранились на диске в разных файлах. Чтобы получить 15-ю запись, надо собрать вместе значения 15-х полей из всех файлов

Запись, относящаяся к Давенпорту, начинается с 1209-го символа. Это число называется указателем записи. Когда Вам потребуется информация, содержащаяся в этой записи, Вы извлекаете указатель из индексного файла и сообщаете операционной системе Вашей микроЭВМ, с какого места на диске считать следующие 93 символа.

Мы упомянули о том, что индекс может состоять из сокращения фамилии. Незачем повторять полностью фамилию "Давенпорт", поскольку она уже хранится в основном файле. Укороченный индекс экономит место и ускоряет процесс поиска. Ключевое поле в каждой записи может содержать, например, начальную букву фамилии, три согласных буквы и указатель ДПРТ1209. Это один из возможных методов "хеширования", очень простой, но не очень практичный (более подробно см. § 2.3).

Организация базы данных с использованием записей фиксированной длины значительно облегчает задачу программиста по написанию СУБД и усложняет жизнь всем остальным. Можно назвать три больших недостатка такой организации.

Очень много места на диске остается незаполненным. Фактически диск будет занят наполовину, поскольку Вы определяете размер каждого поля в записи, исходя из максимально возможной длины элемента, хранящегося в этом поле. И если Вы планировали поместить в базу фамилии вроде "Ханбери–Ханкок–Градислав–Вилсон", а поместили фамилии типа "Хо" или "Чау", почти весь диск окажется пустым. Такое расточительство дисковой памяти нежелательно, так как она все еще остается дорогой. Кроме того, любое число пробелов вряд ли добавляет что-либо к знаниям, накопленным человечеством, но значительно увеличивает размер файла и расстояния, на которые перемещается магнитная головка при поиске.

Время поиска существенно увеличивается. Хорошая программа управления базой данных должна обрабатывать записи переменной длины без каких-либо пропусков. Программа будет сложнее, но зато она обеспечит более эффективное использование поверхности диска. В реальных условиях базы данных приходится реорганизовывать. Спустя несколько месяцев окажется. что под фамилию надо отвести не 15, а 20 символов и что все показатели, хранящиеся в базе, приобретут смысл, если Вы введете еще один дополнительный показатель. Но увы – для него в записи не предусмотрено место. Единственный выход из положения – создать базу данных заново.

Проблема расширения существующих записей становится еще более острой, когда мы приобретаем базу данных, в которой уже хранится информация, но хотим дополнить ее новыми данными (см. гл. 6). Например, предприятие, изготовляющее натяжные гаечные ключи для цепных передач, может приобрести базу данных о названиях и адресах фирм, использующих цепные передачи и потому заинтересованных в покупке специальных гаечных ключей. Эта база данных может быть пополнена информацией, поступающей по телефону от торговых агентов предприятия: "Компания *A* не проявила интереса; компания *B* заявила, что она приобретает 50 штук и ее представитель прибудет в понедельник для переговоров".

Если СУБД не позволяет изменять записи, то предприятие может столкнуться с большими неприятностями. Решение этих и многих других проблем связано с применением записей переменной длины. В пакете программ управления базой данных *Superfile* длина каждого поля определяется метками начала или окончания поля. Признак начала определяет тип хранящейся информации – имя, кредитная ставка, номер телефона и т. д. Признак окончания означает физическую границу поля переменной длины. Так, фамилии и адреса в базе данных, о которой мы уже упоминали, могут храниться на логическом уровне следующим образом:

```
..... (13-я запись)
Имя              = Джон
Имя              = П
Фамилия          = Давенпорт
Адрес            = 16 Линден Др
Адрес            = Оуксайд
Адрес            = Пасадена
Штат             = Калифорния
Почтовый индекс  = CA 12345
(15-я запись) .....
```

На физическом уровне та же запись будет выглядеть так: .... (13-я запись) <ПР1> Джон <КП><ПР1> П <КП><ПР2> ДАВЕНПОРТ <КП><ПР3> 16ЛИНДЕНДР <КП><ПР3> ОУКСАЙД <КП><ПР3> ПАСАДЕНА <КП><ПР4> КАЛИФОРНИЯ <КП><ПР5> CA 12345 <КП><КЗ> (15-я запись) ....

Чтобы сберечь место в базе данных, признаки типа, хранящиеся вместе со словом, имеют размер два байта, что позволяет присваивать им чис-

ловые значения от 0 до 65 000. Им всегда предшествует метка конца поля (КП) или метка конца записи (КЗ), которые имеют значение 253 и 254. Чтобы избежать двусмысленностей, байты данных не должны содержать десятичные коды выше 250 (такое требование не слишком обременительно, поскольку допустимый на ЭВМ набор знаков ограничивается кодом 127). В результате на диске не тратится место на пропуски. Элементы информации имеют переменную длину, которая задается пользователем при вводе информации в тех случаях, когда мы имеем записи типа фамилия—адрес. Поверхность диска используется в два-три раза производительнее. Безусловно, если пользователь пожелает хранить произвольный текст, например комментарии к данным размером от 0 до 5000 символов, единственной приемлемой альтернативой становится база данных с записями переменной длины.

Когда СУБД *Superfile* вносит изменение в запись, программное обеспечение помечает первоначальный вариант как уничтоженный и копирует его вместе с изменениями в базу данных. Изменяться может значение информации, тип которой определяется существующим признаком, или может быть добавлен новый признак и новое значение данных. Таким образом, система манипулирует с записями, форма которых может изменяться, что и требуется в реальных условиях.

2.2. ИНДЕКСИРОВАНИЕ

Все, что хранится на диске — это длинная цепочка байтов, которую операционная система разбивает на файлы. База данных состоит из одного или нескольких файлов. Основная функция программ управления базой даных заключается в том, чтобы найти необходимую запись. Для этого программа использует индекс. В самой идее поиска по индексу нет ничего нового — мы открываем книгу и смотрим указатель, идем в библиотеку и обращаемся к каталогу. Индексы к базе данных служат приближенными описаниями хранящейся в ней информации (эти описания небольшие по размерам и организованы иначе) точно так же, как библиотечный каталог отображает заглавия и авторов книг, хранящихся на полках, а указатель в книге является сокращенным описанием самой книги (можно составить достаточно полное представление о содержании книги, просмотрев указатель). Все они организованы таким образом, чтобы обеспечить быстрый поиск.

Например, указатель к книге состоит из отдельных терминов и номеров страниц, на которых эти термины встречаются. Открыв книгу на нужной странице, Вы просматриваете ее, пока не встретите интересующее Вас слово. Система каталогов в библиотеке обычно включает три каталога: алфавитный, систематический и предметный — и на карточках каждого каталога указывается место нахождения книги на стеллажах, в результате поиск книги ведется на небольшом пространстве книжной полки. Библиотекарь просматривает содержимое полки, пока не найдет подходящую книгу, после чего проверяет ее индекс.

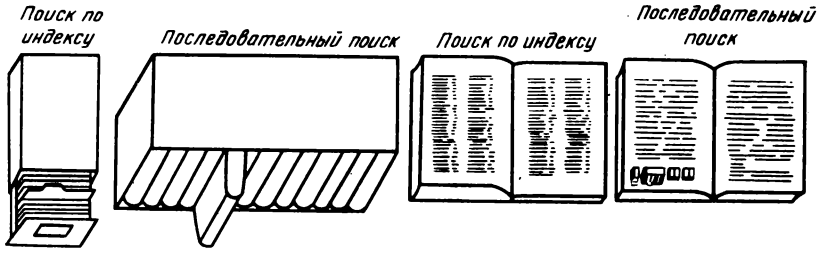


Рис. 18. Поиск по индексу, а затем последовательный поиск позволяют найти в библиотеке нужную книгу

Обратите внимание на то, что последняя операция в индексированном поиске всегда сводится к последовательному поиску. Назначение индекса – сузить зону последовательного поиска.

Работник библиотеки, создавая указатель к книге или организуя каталог, выполняет работу, которую иначе должен был выполнить читатель. Представьте себе, что Вы искатель сокровищ и чтобы найти нить к разгадке захоронения клада, Вам надо просмотреть тысячу книг в частной библиотеке. Вы не знаете, в скольких книгах и где содержится указание на место нахождения клада. Можно раскрыть ближайшую к Вам книгу и начать просматривать книги одну за другой в надежде найти ключ к разгадке сокровищ. Может случиться, что до конца своей жизни Вы будете переходить от полки к полке, бормоча под нос: "Где же я встречал это место о корабле испанской армады?" Разумнее сначала прочитать все книги, завести на них карточки и составить каталог для библиотеки. Тогда Вы сможете проверять различные версии легко и быстро. Это и будет "индексированным поиском". Индекс приведет Вас прямо к цели.

Таким образом, составитель указателя выполняет работу, которую иначе пришлось бы выполнить читателю, чтобы найти в библиотеке интересующие его сведения. Поскольку книга предназначена для чтения многими людьми, ее индексируют еще до создания, чем достигается разумное разделение труда в обществе. То же самое можно сказать и о базах данных. Однако база данных "издается" столькими же людьми, сколькими она используется и здесь разделение труда по индексированию данных не столь очевидно. Надо ли индексировать в момент создания базы данных или лучше сделать это, когда база функционирует? У программиста, разрабатывающего пакет управления базой данных, есть выбор. Можно индексировать каждое слово в записи при вводе его в ЭВМ, а затем столкнуться с ситуацией, когда некоторые индексы останутся без применения, а следовательно, время на их составление окажется просто потерянным. Это означает, что работники, осуществляющие ввод или корректировку данных, должны сидеть и ждать, пока машина индексирует все, что ей предлагают.

Можно собирать новые данные в течение дня, а ночью, когда все расходятся по домам, индексировать их. Или отложить решение вопроса до момента формирования запросов на поиск и заставить программу вести поиск в то время, когда вводятся запросы. Как правило, индексирование стараются осуществить при вводе информации.

Другая проблема, связанная с базами данных, заключается в глубине индексирования. Указатель в книге занимает около 1% объема книги. Однако в вычислительной системе существует возможность гораздо более подробной индексации. Например, очень редко в книгах индексируют денежные суммы. В то же время СУБД должна найти все случаи, когда это значение лежит между 40 и 50 фунтами. В результате индекс получается очень большим. Работая с малыми вычислительными системами, пользователи начинают понимать, что индексы громоздки, если они занимают в памяти больше места, чем сама база.

На практике приходится, как и в любом творческом деле, идти на компромисс. Разработчики индексируют одни слова и оставляют в покое другие, в результате поиск по одним запросам идет быстро, по другим — медленнее. Обычно индексируются отдельные слова, а непрерывный текст не индексируется. Во многих реальных приложениях такой подход оказывается удачным, поскольку индекс получается небольшим, а время реакции систем приемлемым. Часто пользователи могут сказать, что они никогда не будут использовать в качестве поисковых ключей отдельные поля записи, поэтому индекс на оставшиеся элементы информации будет компактнее и просмотреть его можно быстрее.

2.3. ХЕШИРОВАНИЕ

Если Вы хотите прочитать в политической книге о Никсоне, Вы обращаетесь к указателю. Найдя букву "Н", Вы просматриваете содержание, вроде "Нижняя палата в конгрессе", "Никсон, Ричард" и затем начинаете читать внимательно. Индекс содержит то же слово, которое интересует Вас в тексте. В базе данных дело обстоит по-другому.

Чтобы определить, насколько надо выдвинуть магнитную головку, чтобы считать с диска интересующую нас запись, необходимо знать номер записи. Работая с книгой, Вы открываете книжный указатель на первой странице, ищете нужную Вам начальную букву, затем смотрите все подряд. При работе с ЭВМ мы любой ценой хотим избежать сплошного поиска, во время которого магнитная головка устанавливается на начало файла и считывает последовательно все записи, пока не встретит искомую запись. В печатном индексе Вы можете определить необходимую ссылку лишь в том случае, если известно, что ей предшествует. Это можно сделать только путем просмотра. В идеальном случае хотелось бы найти слово "Никсон", ничего не зная о содержании книги или базы данных. Библиотечные работники добиваются этого с помощью десятичной системы классификации Дьюи. Вы просматриваете книгу, определяете ее индекс по десятичной

системе, сверяете его со схемой на плакате, вывешенном рядом с каталогом, и направляетесь прямо к нужной полке стеллажа.

В электронной базе данных Вы делаете перемешивание или "хеширование", т. е. выполняете некоторые действия над словом "Никсон" и преобразуете его в число. Индекс представляет собой таблицу таких чисел и указателей, или адресов, по которым Вы можете найти на диске запись, не обременяя себя поиском каких-либо дополнительных сведений. Процесс преобразования слова в число называется "перемешиванием", вероятно, потому, что при этом первоначальной форме слова уделяется такое же внимание, которое уделяется на мясокомбинате форме говяжей туши при переработке ее на фарш. Существует столько способов хеширования, сколько имеется программистов. Один из способов заключается в следующем: к десятичному коду *ASCII* первой буквы слова, деленному на 10, прибавляется десятичный код второй буквы, деленный на 100, затем третьей буквы, деленный на 1000 и т. д. В результате получается, например, следующее:

Слово	Кодовое обозначение
NIXON	8.62668028447502
ANTIDISESTABLISHMENT	7.372062127717147
ANTIDISESTABLISHMENTARIANISM	7.372062127717147
YRUFGCUSUS	9.812786103029187

С одной стороны, данный алгоритм вполне приемлем поскольку он гарантирует, что любое новое слово будет иметь свое кодовое обозначение. С другой стороны, он страдает тем, что отводит громадные объемы памяти для комбинаций букв, которые не являются словами вообще, как это видно из четвертой записи. Идеальная функция хеширования ставит в соответствие каждому слову, имеющему смысл для пользователя, единственное числовое значение и опускает слова или комбинации букв, не интересующие пользователя. В реальной жизни такая идеальная схема недостижима. Применяемые на практике алгоритмы индексируют слова, которыми никто никогда не пользуется, вызывают конфликтные ситуации, когда два или более слова имеют одинаковые кодовые обозначения, вроде второго и третьего слова в нашем примере. Теоретически этого не должно было случиться, но интерпретатор с языка М-Бейсик, который производил вычисление, не обеспечивает необходимой точности расчетов. В других случаях алгоритм хеширования может не различать все слова, которые по нему обрабатываются.

Поиск функции хеширования – сложный и специализированный процесс, и мы не можем рассмотреть его здесь подробно. Чтобы успешно подобрать функцию хеширования, нужны глубокие познания в области статистики и лингвистики. Кнут [3] отмечает, что только одна из 10 миллио-

нов математических функций, преобразующих слово в число, может быть использована для хеширования. На примере известного парадокса "о дне рождения" Кнут показывает, насколько трудно получать неповторяющиеся индексы. Можно использовать для хеширования фамилии дату дня рождения человека. В году 365 дней, казалось бы, достаточно, чтобы совпадения были редкими. Парадокс заключается в том, что если в комнате собирается не менее 22 человек, то вероятность того, что у двух присутствующих дни рождения совпадают, очень высока.

Зная кодовые обозначения, мы можем определить местоположение записи на диске. Пусть объем диска 10 млн. байт. Если отвести под каждое слово 20 символов, можно разместить на диске 500 000 слов и адрес каждого определить по индексу. Значение индекса выступает в качестве указателя позиции. Самое маленькое кодовое значение имеет буква А — 6.5. Самое большое кодовое значение имеет сочетание ZZZZZZZZZZZZZZZZZ. Оно равно 10.00000010689103.

Теперь можно разделить диск пропорционально этим числам и мгновенно пересчитать индекс слова в его адрес на диске. Казалось бы, все очень просто, но опять мы тратим большие объемы памяти на сочетания, которые никогда не будут использоваться, вроде ZZZZZZZZZZZZZZZZZ или YRUFGCUSUS. Мало вероятно, что в большинстве реальных систем найдет применение слово ANTIDISESTABLISHMENT, но если такое случилось бы, индекс этого слова совпал бы с индексом ANTIDISESTABLISHMENTARIANISM.

Подобные конфликты неизбежно возникают при любой практической схеме хеширования. Выход заключается в том, чтобы ввести в индекс указатель на повторяющийся индекс, в который добавляется при необходимости свой указатель и т. д.

Такая схема индексации очень напоминает схему расстановки книг на полках библиотеки в соответствии с универсальной десятичной классификацией. Скажем, раздел "Вегетация" имеет индекс 631.532/535. Вообразим на минуту, что в государстве издан закон, по которому первая книга о вегетации должна храниться на расстоянии 300 м от начала библиотечного фонда. В этом случае маленькая сельская библиотека должна сооружать длиннющие стеллажи, на которые не хватит книг, в то время как в библиотеках конгресса или Британского музея не хватит места на все книги, относящиеся, например, к "Бактериологии".

Конечно, можно мечтать о такой схеме индексирования, которая могла бы игнорировать содержание остальной части базы данных, но в реальных условиях мы обязаны рассматривать каждое вводимое в базу слово в контексте с другими. Так, в библиотеке нам не обойтись без плаката, показывающего расположение книг на полках и учитывающего все прочие условия. Это ведет к большим затратам времени на индексирование, так как система должна по существу рассортировать базу данных и найти для каждой новой записи свое место, на котором она сочетается с ранее введенными записями. Как мы увидим, дело это весьма трудное.

2.4. ПОСЛЕДОВАТЕЛЬНЫЙ ИНДЕКС

Самый простой способ применения индекса заключается в том, чтобы создать миниатюрную версию базы данных и просматривать ее от начала до конца, когда мы хотим найти что-либо. Пусть в нашей базе данных содержатся сведения о звездах кино. Каждая запись содержит имя, фамилию и краткое пояснение. Для простоты будем считать, что индексируется только фамилия. Результат хеширования фамилии хранится в последовательном файле вместе с указателями *p1, p2* начала каждой записи в файле данных. Чтобы найти "Фифи Лотрек", СУБД хеширует ее фамилию, используя более или менее удачный алгоритм, в значение 7541 (оставим в стороне вопрос о том, почему "Лотрек" превращается именно в 7541). Затем программа просматривает индексный файл, пока не обнаружит число 7541. Теперь она загружает адрес *p2* и отыскивает по нему запись в базе данных. Программа проверит, содержит ли действительно эта запись слово "Лотрек", как указывалось в запросе. Такая проверка необходима, поскольку любая реальная система хеширования, ориентированная на экономию памяти, неизбежно выдаст одни и те же кодовые обозначения для нескольких различных слов. Фамилия "Лотрек" преобразуется в число 7541, но тот же результат может дать фамилия "Констанца", поэтому программа должна осуществить проверку, прежде чем передать пользователю нужную запись.

Любая операция, выполняемая в базе данных, прежде всего характеризуется своей продолжительностью. На просмотр индекса уходит время. Операция поиска в индексе выглядит очень простой, но то, что происходит фактически, оказывается намного сложнее.

Прежде чем начать просматривать индексный файл, его нужно переписать с диска в оперативную память. Магнитная головка устанавливается на начало файла и считывает часть файла (весь файл слишком велик, чтобы вместиться в оперативную память) в так называемый буфер или зарезервированную область памяти. Программа просматривает содержимое буфера, осуществляя поиск первой цифры кодового обозначения 7. Если первая цифра индекса не 7, программа переходит к следующему закодированному слову, так как нет смысла проверять остальные три символа. Когда начальная 7 найдена, программа проверяет среднюю цифру, равна ли она 5. Если равна, то проверяется следующая цифра, если нет, программа переходит к новому слову. И так до тех пор пока не будет прочитано все содержимое буфера, после чего в буфер считывается следующая часть файла и весь процесс повторяется.

Вообще говоря, операции в памяти выполняются очень быстро. Время уходит на перемещение магнитной головки вдоль поверхности диска и считывание порций файла в буфер. Вот почему чем больше буфер, тем меньше выполняется операций ввода-вывода и быстрее ведется поиск по индексу.

Еще одна операция, требующая затрат времени — верификация или, другими словами, проверка записи с соответствующим кодовым обозначе-

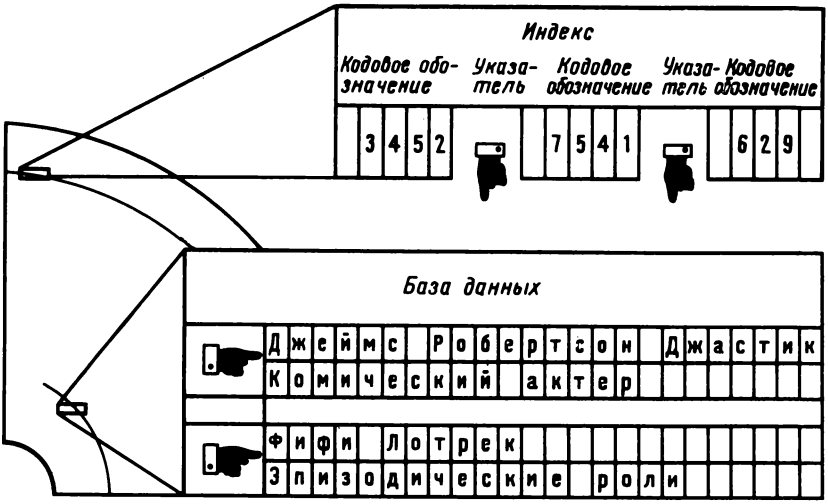


Рис. 19

нием на то, что она действительно содержит нужную фамилию. Верификация связана с операциями ввода-вывода и также требует времени. Чем грубее индексирование, тем больше тратится времени на верификацию. Приведем пример. Допустим в базе данных хранится информация о 100 000 актеров. Алгоритм хеширования выдает четыре десятичные цифры, поэтому в индексе всего может быть 9999 кодовых обозначений. Это значит, что в среднем каждое обозначение повторится в индексе 10 раз. В первом приближении получается, что поиск по известной фамилии в одном единственном файле требует около десяти проверок. Однако фактически их число будет меньше. Как только фамилия будет найдена, поиск прекращается. С одинаковой вероятностью может искомая фамилия оказаться как в конце файла, так и в начале. Поэтому в среднем число верификаций не превысит пяти. Чем больше глубина индексирования, а она определяется разрядностью кода, тем меньше проверок. Если длину кода увеличить на одну цифру, то число совпадений уменьшится в 10 раз, а число проверок – в 5 раз. Соответствующим образом сократится время поиска, однако размер индексного файла увеличится на 25%. В нем возрастет число указателей. На одну четверть увеличится число операций ввода-вывода при считывании содержимого файла в буфер. Все это снова приведет к возрастанию времени поиска. Индексный файл может оказаться непропорционально большим по отношению к самой базе данных. Правда, учитывая, что стоимость дисковой памяти постоянно снижается, последнее обстоятельство не имеет особого значения.

Существенный недостаток поиска простым перебором в дебрях последовательного индекса заключается в том, что приходится просматривать весь индекс, чтобы извлечь требуемую запись.

Однако можно поступить по-другому. Когда Вы ищете номер телефона в телефонном справочнике, то не листаете огромный том с первой страницы, а открываете его примерно в том месте, где находится начальная буква слова, которое Вас интересует. Допустим, Вы ищете номер телефона Норвежского центра технического перевода. Вы пропускаете желтую часть справочника (буквы А–Д), розовую (буквы Е–К) и раскрываете несколько раз зеленую (буквы Л–Р), пока не увидите напечатанные жирным шрифтом вверху страницы буквы Н и затем Но. Теперь можно читать текст, набранный мелким шрифтом.

Рациональная организация телефонных справочников подсказывает нам, когда надо приступать к последовательному поиску, который с какого-то момента времени становится неизбежным. Секрет быстрого поиска в том, чтобы начать последовательный поиск как можно позже.

Так же как разделена телефонная книга, можно разделить индекс, и тем самым ускорить в нем поиск кодового обозначения 7541. Несложно разделить индексный файл на 10 меньших файлов, по одному на каждую начальную цифру. Учитывая, что кодовое значение начинается с цифры 7, мы будем хранить число 541 в седьмом файле. Размер файла, безусловно, уменьшится, так как мы экономим по одной цифре для каждого кодового обозначения. Но надо добавить обозначение файла и указатель, в результате выигрыш получается иллюзорным. Настоящий выигрыш связан с повышением быстродействия поиска. При организации поиска в разделенном индексе не надо искать число 7541, мы сразу обращаемся к среднему файлу и ищем число 541. Не надо тратить время на просмотр других девяти файлов, сокращается число операций доступа к диску, уменьшается на 25% длина кода, меньше цифр приходится сравнивать при поиске.

Мы можем продолжить процесс разделения индекса дальше, применив тот же принцип ко второй, третьей и четвертой цифрам кодового обозначения.

Время поиска уменьшается, но память будет использоваться нерационально. На первом уровне индексирования мы имели лишь десять обозначений файлов, соответствующих начальным цифрам 0, 1, 2, ... 9 в четырехзначном индексе, но каждое из них отсылает к 100 обозначениям на втором уровне, 1000 на третьем и 10000 на четвертом, что приводит к одинаковым размерам с последовательным индексным файлом (точнее на 11% больше первоначального индекса плюс указатели).

Однако времени на поиск потребуется значительно меньше, так как достаточно четырех обращений к диску, чтобы ЭВМ прочла значение адреса записи в базе данных.

В начальный период развития баз данных организация файлов сильно зависела от природы устройств внешней памяти, и иерархия индексов

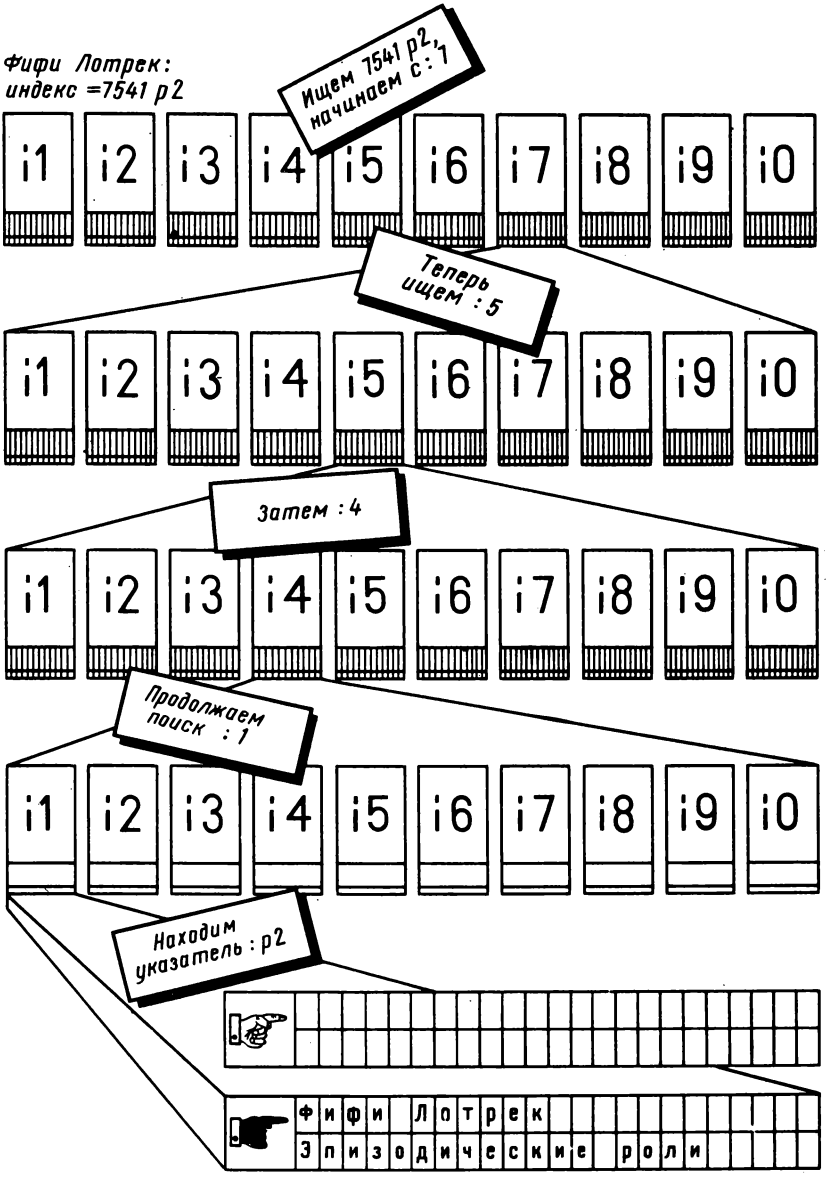


Рис. 20. В основе индексно-последовательного метода доступа лежит следующий принцип: индекс на каждом уровне ведет к 10 дополнительным индексам, независимо от того, использованы они при поиске или нет

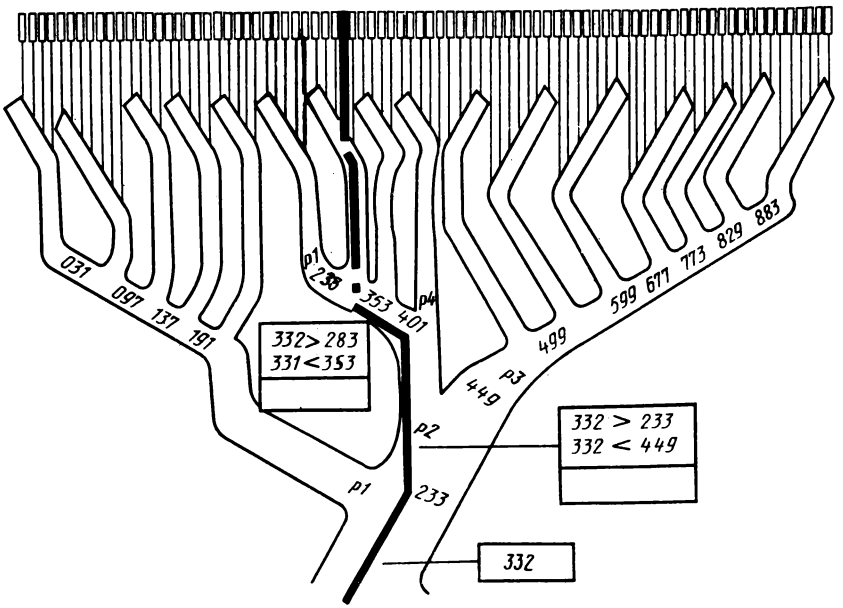


Рис. 21. Организация индексов в виде B-деревьев позволяет лучше использовать поверхность диска и обеспечивает более быстрый поиск, чем ISAM

строилась на иерархии дорожек, секторов, номеров записей данных, хранимых на диске. Такой метод обработки файлов назывался индексно-последовательным, сокращенно *ISAM*. Сегодня, когда специалисты употребляют этот термин, они большей частью имеют в виду многоуровневый метод индексирования с использованием записей постоянной длины. Мы можем рассматривать первый, второй и третий уровни как индексы к четвертому уровню, применение которых существенно ускоряет поиск интересующей нас записи. Поскольку небольшое число записей на верхнем уровне связывается со множеством записей на нижнем уровне, такая организация данных называется деревом, хотя оно изображается в перевернутом виде. От каждой записи отходят воображаемые ветви, образуя узел дерева. Если продолжить сравнение с деревом, то концевые вершины, содержащие прямые ссылки на базу данных, называются листьями .

Такой метод организации индекса не является самым гибким и эффективным. В реальной базе данных нам потребуются не все возможные 10 000 кодовых обозначений – часть места будет потрачена впустую. С другой стороны, функция хеширования способна породить миллионы кодовых обозначений, и только часть из них будет использована. Таким путем невозможно обеспечить место для хранения всех записей, полученных по алгоритму хеширования.

В развитых базах данных для файлов индексов строится так называемое *B*-дерево или симметричное, сбалансированное дерево, у которого все пути от корня к вершине имеют одинаковую длину. Такая организация файлов обладает большой гибкостью при сохранении фундаментальной структуры.

На каждом уровне или в каждом узле *B*-дерева содержится несколько кодовых обозначений и указателей. Кодовые обозначения располагаются слева направо в порядке возрастания их значений.

Для того чтобы найти лист, содержащий требуемый индекс, например 331, начнем поиск с корня дерева. Корень содержит: *p1*, 233; *p2*, 449, *p3*. Число 331 больше, чем 233, поэтому мы пропускаем указатель *p1* и выбираем указатель *p2*, так как наше число меньше 449. (Указатель *p1*, *p2* и т. д. безусловно различные на каждом уровне).

Указатель *p2* указывает на узел, содержащий запись: *p1*, 283, *p2*, 353, *p3*, 401, *p4*. Число 331 лежит между 283 и 353, поэтому мы снова выбираем указатель *p2*. Поиск продолжается до тех пор, пока мы не извлечем из концевой вершины адрес записи в основном файле базы данных.

Следует обратить внимание на то, что узлы *B*-дерева редко бывают заполнены до конца. Если Вы хотите добавить новый индекс с указателем на базу данных, Вы помещаете его в соответствующий узел самого нижнего уровня. При заполнении узла Вы делите его на два наполовину заполненных узла и вставляете дополнительный указатель в узел на следующем уровне, с которым они связаны. Если и этот узел заполняется полностью, Вы также делите его пополам и вставляете указатель в узел на следующем уровне и т. д. Обо всем этом можно подробно прочитать в книге Кнута [3], но эти сведения нужны в основном системным программистам, которые рассматривают данные как цепочку символов безотносительно к их содержанию. Задача программистов – записать и найти такую цепочку как можно быстрее. Конечного пользователя база данных интересует как средство получения необходимых ему сведений и цепочка символов для него – это число (цена свинины на рынке) или фрагмент текста (отчет о крушении испанского корабля с сокровищами или кадровые сведения о персонале). При автоматическом индексировании программы по-разному индексируют числа и текст. Обычно числа индексируются таким образом, что кодовые обозначения, скажем для 1 или 2, не слишком отличаются друг от друга, в то время как для числа 999 999 999 кодовое обозначение совсем иное.

Другие проблемы возникают при нахождении индексов слов: пользователю может потребоваться текст, начинающийся с буквы А или содержащий слово "спиртное" (если Вы думаете о том, кого можно командировать в Саудовскую Аравию). Можно предположить, что каждое поле данных содержит одно слово и индексировать его. Но практически во многих случаях поля содержат несколько слов. Один из выходов заключается в том, чтобы индексировать первое из существенных слов, если таковое имеется. Тогда индексом для А.К. Джонс, Доктор Джонс, Господин Джонс, Сэр Джонс будет Джонс. Разработчики баз данных обычно выде-

ляют инициалы и титулы в отдельные поля с тем, чтобы индексаторы могли иметь дело только с одним именованным полем "фамилия". Еще сложнее обстоит дело с индексированием последовательных текстов, таких как записи в системах регистрации медицинской информации, выпуски новостей, краткие изложения книг или газетных статей в информационно-справочных системах.

Индексирование последовательного текста связано с решением двух взаимосвязанных проблем: какая информация в действительности нужна пользователю и как найти место для ее хранения. Было бы неправильно подходить к индексированию текста так же, как к индексированию значений отдельных полей записи. В этом случае этот абзац будет индексирован по первому существенному для понимания его смысла слову: "проблемы". На первый взгляд вроде неплохо, но вряд ли Вам поможет индекс в конце книги, где будет записано "Проблемы, 56". Проблемы в чем? Бесполезно также индексировать каждое слово в абзаце. Представьте себе базу данных, в которой индексированы все служебные слова. Конечно, этого делать не надо, и интересующие нас слова встречаются гораздо реже других слов, но по какому критерию их отбирать? Если мы условимся, что вероятность появления в тексте коротких служебных слов превышает X % и поэтому надо индексировать слова, которые встречаются реже, то как быть со словом "поэтому", индексировать которое тоже не имеет смысла? Очевидно, надо оказать помощь индексатору и попытаться автоматизировать процесс. Он, а чаще она, просматривает текст и отмечает слова, которые могут интересовать читателя. Затем ЭВМ хеширует отмеченные слова и помещает кодовые обозначения в B-дерево вместе с указателями записи, в которой содержится сам текст. Возможен вариант, когда индексатор заранее составляет список разрешенных ключевых слов. СУБД затем отыскивает в заданном тексте ключевые слова. Например, нефтедобывающая компания может использовать базу данных о своих многочисленных эксплуатационных буровых скважинах. Запись о каждом объекте содержит описание геологической структуры в зоне скважины. Прежде чем пробурить первую скважину и ввести информацию о ней в базу данных, разработчики системы уже решили, что термины, подобные терминам "лигнит", "оолит", "падение пласта", необходимо индексировать. Применение списка разрешенных ключевых слов возможно, если пользователи уверены в том, что слова, которые они ищут, индексированы, не говоря уже о том, что эти слова вообще есть в базе данных.

Можно попытаться поручить ЭВМ отбирать ключевые слова. Единственно обнадеживающий пока способ заключается в том, чтобы определить частоту появления в тексте каждого слова и сравнить со средней частотой, с которой это слово встречается в обычном тексте. Ключевые слова чаще повторяются в тексте, чем обычные слова и могут быть определены автоматически, однако затем они должны быть отредактированы индексатором.

Мы можем сделать в базе данных агентства печати запрос на статью, содержащую ключевые слова: "волнения", "посольство", "ребенок",

"перелом", "нога", "моряк", в которой описана история о том, как военный моряк спас во время возникших волнений и беспорядка маленькую девочку. Он доставил ее в посольство, где врачи оказали ей помощь, так как у нее была сломана нога. Мы можем исчерпать весь список разрешенных ключевых слов и не установить при поиске нужного соответствия, поскольку в статье говорится о травме, а не о переломе ноги. Следовательно, после того как мы тем или иным способом индексируем текст с помощью ключевых слов, СУБД должна при неудачном поиске повторить его уже без ключевых слов и получить хотя бы одно совпадение. В идеальном случае необходимо использовать некий тезаурус, который бы различал синонимы, тогда система могла бы осуществлять по ним поиск. Однако это дело будущего.

2.6. СОРТИРОВКА

Разрабатывать программное обеспечение управления базами данных было бы намного легче, если бы пользователи меньше увлекались сортировками. Сортировка обеспечивает упорядочение при печати искомых записей по возрастанию или убыванию значений отдельных полей.

Например:

Альберт Албертсон	Альфа, 12	Атабаска
Джон Албертсон	Альфа, 12	Атабаска
Сусанна Брайент	Альфа, 2	Атабаска
Артур Аск	Бенсон авеню, 223	Атабаска
...	...	...
Герберт Зелит	Аннеймд др., 1	Уошоу

Примечание. Поля, расположенные справа, играют роль ключей, по которым осуществлялась сортировка.

Необходимо различать понятия "сортировка" и "выборка". Телефонная книга рассортирована по фамилиям, без этого ею нельзя было бы пользоваться. Однако если содержимое справочника ввести в базу данных, то записи становятся доступными не только по фамилии, но и по названию улицы, номеру телефона. В базах данных нет необходимости что-либо сортировать, так как поисковый механизм обеспечивает динамическую выборку данных по любому признаку. Результаты сортировки, как правило, выводятся на печать. Можно надеяться, что по мере перехода к безбумажной технологии сортировка также исчезнет. Почему автор так настроен против сортировки? Главным образом потому, что она отнимает много времени. Попробуйте рассортировать игральные карты по масти, а внутри масти – в восходящем порядке их значений. Каждый раз, снимая карту сверху нерассортированной колоды, Вы вынуждены просматривать уже рассортированные карты, чтобы вставить новую карту в надлежащее место. На это уходит время. При сортировке ЭВМ поступает таким же обра-

зом, выполняя многочисленные операции поиска, прежде чем включить запись в сортируемый файл. "У меня на руках тройка червей, – где у нас черви? Где двойка или четверка? Не могу найти. Ну, ладно, а где туз или пятерка? ..."

В колоде небольшое, заранее известное число карт, и можно обозначить на листе ватмана место для каждой карты, разложить их по этой схеме, а затем собрать снова в колоду. Но с базами данных дело обстоит не так просто. Например, Вам надо расположить записи, содержащие имена, фамилии и адреса, в алфавите фамилий, названий улиц и городов. Вы не знаете заранее место записи в создаваемой последовательности. Позиция каждого адреса зависит от любого другого адреса. Простейшие алгоритмы сортировки требуют сравнить каждый адрес со всеми другими адресами.

Если имеется n адресов, то число сравнений пропорционально n^2. Сортировка таким способом нескольких сотен адресов, находящихся в оперативной памяти, может быть осуществлена достаточно быстро. Но если надо рассортировать несколько тысяч или даже сотен тысяч адресов, то они должны храниться на диске. Каждая операция сравнения требует считать с диска два адреса в память ЭВМ, сравнить их значения и переместить записи с одного места на другое.

Предположим, что надо рассортировать 1000 адресов. Время обращения к диску – для того чтобы считать адрес в оперативную память – 0,033 с (приемлемое значение для твердого диска небольших размеров). Весь процесс сортировки займет $1000 \times 1000 \times 0{,}066$ с (без учета времени, затрачиваемого ЭВМ на сравнения в оперативной памяти) или 16,66 ч.

Время, необходимое для сортировки десяти тысяч адресов, приобретает уже фантастические размеры. Поскольку операции сортировки часто используются в задачах обработки данных, найдены весьма эффективные методы сортировки, при которых теоретически число сравнений равно $n \log n$, если сортируется n объектов. Возможно для нематематика это мало что говорит, но выигрыш во времени получается значительный: 10 000 адресов могут быть рассортированы такой программой за 1,5 ч. Написать эффективную программу сортировки сложно, но соответствующие алгоритмы изложены во многих книгах.

Из базы данных можно извлекать записи, упорядоченные по многим признакам. Допустим, Вам нужен список покупателей, рассортированный по фамилиям, названиям городов и районов, а также по возрасту и размерам задолженности. Очевидно, физический файл не может быть упорядочен по двум признакам одновременно (также как нельзя сразу разложить колоду карт по масти и отобрать четыре туза, четыре короля и т. д.). Сортируемыми файлами становятся индексы. Используя упорядоченные индексы и указатели, можно извлечь из базы данных записи в любом порядке. В некоторых СУБД предусматривается постоянная сортировка файлов. Преимущество такого подхода состоит в том, что Вы можете просмотреть список своих клиентов, например в порядке возрастания номеров сопроводительных документов на отпущенные им товары. Недостаток такого подхода проявляется при добавлении новых или корректировке

существующих записей, так как приходится тратить время на поиск их места в рассортированном файле. Оператору приходится тратить время на ожидание. Кроме того, файл может быть заранее упорядочен лишь по одному признаку.

ГЛАВА 3
ИНТЕРФЕЙС КОНЕЧНОГО ПОЛЬЗОВАТЕЛЯ

Мы лишь поверхностно рассмотрели ту часть работы пакета программ управления базой данных, которую фактически наблюдает пользователь.

Все, что было сказано об организации файлов, очень важно, однако не об этом в первую очередь думает служащий, когда в понедельник утром приходит на работу и видит перед собой кипу заданий, которые надо ввести в ЭВМ.

Программное обеспечение базы данных имеет два уровня. На нижнем уровне выполняются операции по хранению и поиску данных. На верхнем уровне данные вводятся в базу, проверяются, над ними производят расчеты и результаты выводятся на печать. В предыдущей главе мы бегло познакомились с формированием, поиском и выдачей данных. Теперь в центре нашего внимания будет работа программ второго уровня, обеспечивающих интерфейс пользователя с базой данных. Именно работу этих программ наблюдает пользователь.

3.1. ФОРМЫ И ИХ ЗАПОЛНЕНИЕ

Первое, что Вы должны сделать, загрузив в память ЭВМ пакет программ управления базой данных – начать вводить записи. Проще всего это сделать, заполняя на экране дисплея какую-либо форму. Форму проектирует пользователь, на экран ее выводит программа, называемая генератором форм.

Форма может не отличаться от бланка документа, который она заменяет. Надо только помнить, что данные вводятся в форму последовательно. Форма записывается один раз и хранится в памяти машины. Ввод и вывод информации из базы происходит с помощью формы. В процессе ввода создается "плоский файл", в котором записи имеют одинаковый формат, в отличие от файлов в базах данных с иерархической или реляционной структурой (см. гл. 4).

Форма обычно включает поля и текст. В поля Вы вводите их значение, текст содержит примечания, инструкции и т. п. Опытные программисты часто выделяют эти два элемента формы различными способами: шрифтом, цветом, нормальным или инверсным изображением. Выбор способа не имеет решающего значения.

На первый взгляд может показаться, что именованные поля форм, подсказывающие Вам, какую информацию надо вводить с клавиатуры, ничем не отличаются от именованных полей записи, описанных в гл. 2.

Часто так оно и бывает: Вы вводите название фирмы в поле, именованное "Компания", и оно хранится как элемент записи с именем "Компания". Однако это необязательно. Допустим, эмиграционная служба может использовать форму, в которой запрашивается девичья фамилия матери эмигранта. Однако в базе данных эта фамилия будет храниться в поле с именем "Фамилия",так же как и любые другие фамилии.

Если в базе данных используются записи постоянной длины, то в форме на экране размеры данных обозначаются соответствующим количеством точек или черточек, по одной на каждый будущий символ. Точки или черточки исчезают по мере ввода информации. Можно указать длину с помощью скобок, отнесенных друг от друга на заданное расстояние.

В тот момент, когда Вы вводите данные, удобно проверить, что именно Вы ввели. Сложность работы с ЭВМ заключается в том, что информация в них скрыта от пользователя, а обработка ее происходит очень быстро. Если Вы ошиблись при вводе данных, то Вы можете не знать об этом, пока не получите жалобу от клиента, в которой он сообщает, что ему выставили счет на 9 999 999,99 фунтов. Поэтому хороший генератор форм должен не только принять данные и передать их программам формирования базы данных,но и выполнить ряд проверок.

Проверки могут быть самыми различными, но обычно они сводятся к следующим:

1) поиск в словаре. Допустим, Вы работаете агентом страховой компании и страхуете по полису, выпущенному компанией "Стандарт Лайф оф Канада". У Вас нет желания писать каждый раз полное название компании и адрес потому, что это случается много раз за день. Вы пишете сокращенно "СЛК", а СУБД находит название и адрес и записывает их в запись базы данных. Все это выглядит просто, однако возникают вопросы, связанные с реляционными базами данных, которые мы рассмотрим в гл. 4;

2) верификация. Проверку любых данных можно осуществить, заставив оператора дважды вводить одну и ту же информацию. После первого набора данных на клавиатуре и отображения их на экране экран очищается и оператор повторяет ввод данных. При несовпадении запись аннулируется. Этот метод проверки позволяет выявить ошибки, допущенные оператором из-за невнимательности, но вызывает некоторое раздражение со стороны персонала. Возможно, Вы используете этот метод в исключительных случаях, например при вводе номеров страховых полисов;

3) проверка диапазона значений. Программа должна проверить, лежит ли вводимое число в заданном диапазоне значений, установленном при создании формы. Например, счет за израсходованную электроэнергию может составлять от 0 до 1000 фунтов. Если оператор по рассеянности введет 34567 вместо 345.67, ошибка будет обнаружена программой;

4) проверка по списку. Аналогичным образом проверяется текстовая информация. Для этой цели используется список, с которым сверяются элементы текста. Допустим, в базе данных накапливается информация о техническом обслуживании колонны грузовых автомобилей. Каждый раз,

Имя поля		Значение
Имя	=	Джон
Имя	=	П
Фамилия	=	Давенпорт
Адрес	=	16 Линден Др
Адрес	=	Оуксайд
Адрес	=	Пасадена
Почтовый индекс	=	CA 12345
Телефон	=	01234567
Кредит	=	5000
Примечания	=	Потребность в месяц < 1000 Оплата 60 дней
Дата	=	5 11 84
Стоимость	=	869.00
Остаток	=	1381.00
Возврат	=	121.00
Баланс	=	2129.00

а)

а Имя: [Джон] [П]
Фамилия: [Давенпорт]
Телефон: [01234567]

б Адрес: [16 Линден Др]
[Оуксайд]
[Пасадена]

в Индекс: [CA 12345]

г Лимит на кредит: [5000]

д Примечания: [Потребность в месяц < 1000.
Оплата 60 дней]
[ ]

е Дата; [5 11 84]

ж Стоимость: [869.00]
Остаток: [1381.00]
Возврат: [121.00]

з Баланс: [2129.00]

б)

Рис. 22. Структура записи в базе данных Superfile (а) и ее отображение на экране (б). Запись имеет простой формат и позволяет людям, машинам и программам взаимодействовать между собой. Она состоит из именованных полей (слева) и их значений (справа). Вместе они являются описанием объекта. Новые элементы информации могут быть добавлены в любое время. В базе данных хранятся записи различных типов. Обычно записи отражаются на экране в виде заполненной формы:
а, б – повторяющиеся поля с именами Имя и Адрес; *в* – поле, содержащее почтовый индекс; *г* – лимит на кредит, значение от 10 000 до 100; *д* – символьное поле для примечаний; *е* – автоматический перевод даты в номер дня, начиная с 1 января 0 г. н. э.; *ж* – денежные суммы, допустимые значения до 1000; *з* – поле для записи результатов расчетов (стоимость складывается с остатком и вычитается возврат кредита)

когда автомобиль проходит техническое обслуживание, в базе данных появляется запись, включающая его регистрационный номер. Было бы неплохо каждый раз уточнять, что это номер одной из машин колонны, а не номер "самозванного" грузовика.

Программа должна выбрать значение соответствующего поля и проверить, встречается ли оно где-либо в базе данных. Полезным вариантом этого способа является ввод первых нескольких букв хранящегося текста и затем вывод на экран полного текста — результат получается такой же, как при поиске в словаре.

Обратная проверка также полезна: удалить то, что уже находится в базе данных. Это нужно, например, при вводе номеров страховых полисов, чтобы исключить возможность совпадения номеров.

Часто в записи имеются поля, значения которых можно считывать, но нельзя изменить. В этом случае программа должна выполнить функцию блокировки полей записи, которая не позволит пользователю ввести или изменить данные. Вся форма может быть защищена от записи. Даты также необходимо проверять, кроме того, их нередко приходится пересчитывать в "номер дня". Не так просто машине подсчитать, скажем, день платежа по накладной, если оплата намечена через 30 дней после выписки накладной. Часто с датами проводятся еще более сложные расчеты и проще всего их выполнить, если перевести каждую дату в число дней, прошедших от некоторой произвольно выбранной точки отсчета, например 1 января 0 года нашей эры.

Поскольку при этом надо сообщить программе, что в феврале бывает 29 дней в високосные годы, если только год не делится на 100, то добавив еще некоторые условия, можно убедиться, что дата введена правильно во всех отношениях.

3.2. СОХРАННОСТЬ ИНФОРМАЦИИ В ЭВМ

Сохранность информации прежде всего связана с обеспечением физической целостности данных.

Эти вопросы были рассмотрены в гл. 1. Менее важным, по крайней мере для микроЭВМ, является ограничение доступа к информации лицам, не имеющим на то специального разрешения. Возможно нарушители понимают, что машины настолько ненадежны, что в них вряд ли можно найти что-нибудь полезное после попытки проникнуть в них. Из собственного опыта, связанного с продажей пакетов управления базами данных некоторым высокопоставленным лицам и многим рядовым гражданам могу сказать, что сохранность данных в этом смысле ни у кого не вызывает особых проблем.

Но, возможно, такая проблема еще возникнет и лучше поговорить о ней заранее, поскольку закон о защите данных, принятый в Англии, квалифицирует как уголовно наказуемый проступок хранение персональной информации в общедоступных местах.

В основе методов, контролирующих доступ к системе в целом или к отдельным элементам информации, лежат пароли.

Когда Вы загружаете систему, машина просит Вас ввести пароль. Вы набираете пароль на клавиатуре и впервые на экране ничего не появляется. Опыт показывает, что очень трудно из-за спины оператора прочесть слово, набранное на клавиатуре, и, конечно, если на экране ничего нет, любопытствующему не удается что-либо узнать. Системы с паролями могут быть довольно сложными. Эффективной является система с несколькими уровнями защиты данных, например четырьмя уровнями, ограничивающими доступ к записи целиком или к ее отдельным полям. Номера уровней защиты данных и номера уровней допуска к ним взаимосвязаны. Тот, кто имеет допуск четвертого уровня, может пользоваться любыми данными. Лица с допуском третьего уровня могут пользоваться данными с уровнями защиты 1, 2 и 3. Обычно применяют две системы защиты. Одна контролирует чтение данных, другая – запись. Работнику может разрешаться чтение определенной информации, но ему запрещается пополнять или изменять ее. С другой стороны, тому же работнику может быть разрешено вносить данные, но запрещено читать данные из одного и того же файла. В больших реляционных базах данных разграничение доступа к данным может оказаться удивительно запутанным.

Уровень допуска определяется паролем. Безусловно в базах данных с системой защиты информации должны использоваться методы криптографии, иначе через операционную систему можно распечатать содержимое основных файлов базы данных.

В системе защиты базы данных ключевой фигурой является "Администратор" базы данных – лицо, которое владеет паролем ко всей системе и контролирует ее работу. Если администратор, не надеясь на свою память, запишет пароль к системе на стене у терминала, то могут начаться крупные неприятности.

Для больших ЭВМ, особенно тех, которые подключены к общим телефонным сетям, вопросы защиты информации перерастают в настоящую проблему.

В газетах часто сообщают об изобретательных подростках, которые подключаются к компьютерам в вычислительных центрах банков и открывают себе кредит на триллионы долларов или низвергают мир в пучину мировой войны и т. д. К счастью, для пользователей микроЭВМ эта проблема имеет меньшие масштабы, поскольку физический доступ к ЭВМ ограничен доступом в помещение, где они находятся, или в подразделение, которое их использует.

3.3. ВРЕМЯ РЕАКЦИИ СИСТЕМЫ

Весь смысл организации базы данных на ЭВМ вместо бумажных картотек заключается в возможности осуществить поиск быстро и по различным условиям. Если СУБД не может этого делать, она не выполняет свои функции. Но как определить, достаточно ли быстро осуществляется поиск? Все зависит от того, кто ждет ответа от системы и почему.

Представим себе, что базу данных используют в своей работе операторы-телефонистки. Люди часто звонят, чтобы заказать товары, или узнать, вовремя ли прибывает самолет, или интересуются, не объявлен ли розыск на человека, которого они встретили на улице (в этом случае эти люди – полицейские). Оператор сидит перед терминалом и ждет, когда можно прочитать ответ. Подобные системы называются интерактивными (имеется в виду, что когда Вы обращаетесь к ним с вопросом, то получаете ответ). Предполагается, что время обработки запроса составляет обычно 1–2 с. Опыт показывает, что более длительное ожидание плохо действует на нервную систему человека, а время обработки запроса 10 с оказывается недопустимым. Таким образом, для интерактивных систем существует предельное время обработки оперативных запросов.

Конечно, многие приложения в базах данных не требуют такой оперативности. Если Вы распечатываете почтовые ярлыки для рассылки корреспонденции или платежные извещения (на профессиональном жаргоне такая обработка называется пакетной), то можете оставить машину работать на ночь и время обработки для Вас не имеет значения. Часто для вывода записи на печать требуется больше времени, чем на ее поиск. В этом случае время обработки запроса не имеет значения.

Однако СУБД никогда не может быть слишком быстрой. Пока мы вели речь о поиске одиночных значений. Когда прибывает самолет рейса 356 из Аммана? Такой запрос требует поиска по одному условию. Вы вводите номер рейса и видите на экране запись, содержащую самую свежую информацию о выполненном рейсе. К сожалению, базы данных решают более сложные задачи. Даже при пакетном решении нам может потребоваться многоаспектный поиск. Пакет программ управления базой данных, не обеспечивающий малого времени обработки запроса, может катастрофически затянуть обработку информации.

Базы данных могут принести еще большую пользу в будущем, когда наряду с цифрами и текстом они будут хранить изображения и осуществлять их поиск. На рисунке показано, как через несколько лет графические базы данных могут помочь в борьбе с преступностью.

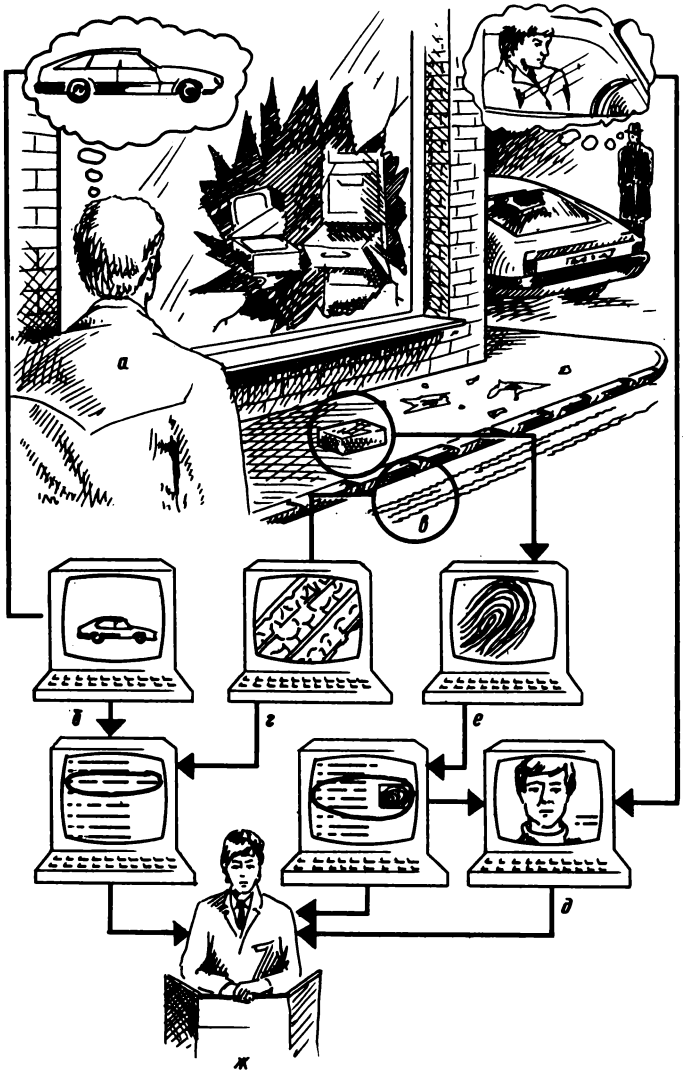


Рис. 23. Свидетель ограбления видит отъезжающий автомобиль (а). Зная модель автомобиля и его цвет, мы получаем доступ к записям в базе данных государственной автоинспекции (б). Полученные на месте преступления отпечатки протектора (в) позволяют вести поиск в базе данных промышленной продукции (г), что сужает круг разыскиваемых автомашин (д). Второй свидетель видит водителя. Составляется фоторобот, используя который, в базе данных уголовного розыска отбирают записи на нескольких преступников. Частичный отпечаток пальцев на кирпиче, которым преступник разбил витрину, позволяет уменьшить число отобранных записей (е). Изучение записей, полученных из трех баз данных, вывело следствие на 2–3 лица, подозреваемых в преступлении. Вскоре преступника арестовывают (ж)

3.4. ПОИСК В БАЗЕ ДАННЫХ

Как в базе данных организован поиск информации? Вы вводите фамилию и получаете адрес. Вы знаете точно, как пишется фамилия и программа находит ее в базе данных. Поиск ведется, как говорят, до "полного совпадения". Вы можете ввести название города и получить список его жителей.

Однако очень часто Вы не знаете, с чего начать. Вам нужны методы "приближенного поиска". Человек, которого Вы разыскиваете, живет в районе Крисент, но Вам достаточно набрать на клавиатуре *Крис*. Почему? Да потому, что с молчаливого согласия программистов всех стран знак * означает соответствие любому сочетанию символов. В этом случае вместо Крисент может быть использована аббревиатура Крис. Такому условию поиска будут соответствовать Морнингтон Крисент или Юнипер Крис, но, к сожалению, и "Кристалл". Надо быть внимательным при составлении запроса. Но даже если Вы внимательны, база данных не один раз может удивить Вас.

Если Вы собираетесь поехать в Рединг и хотите иметь список всех проживающих там клиентов, Вы сможете ввести "РГ*" в поле почтового индекса и программа найдет Вам РГ 37КТ или РГ 01ВБ3. Если Вы гражданин США и хотите написать письма всем знакомым, проживающим в Монтане, Вы организуете поиск по условию "Почтовый индекс =МО*".

Используя этот же способ, можно искать отдельное слово или сочетание слов в тексте. Например, при поиске в базе данных отдела кадров кандидатов для направления на работу в арабские страны в качестве критерия поиска можно использовать слово "* спирт *" в поле комментариев. Этому критерию удовлетворяет, например, комментарий: "Приятный человек, но, к сожалению, злоупотребляет спиртными напитками в свободное от работы время".

Реже возникает необходимость найти слово, в котором известно число букв, но сами буквы известны не все. Допустим, Вы хотите в запросе указать имя, но не помните точно "ТОМ", "ТИМ" или "ТАМ". Можно осуществить поиск по команде "Т??", в которой каждый вопросительный знак означает любой одиночный символ. Предположим в базе данных хранится информация о запасных частях к двигателям, причем каждая деталь имеет код вида ДКЕ 326. Вам известно, что коды деталей дизельных двигателей начинаются с буквы Д, а детали выхлопной системы имеют букву Е в третьей позиции. Чтобы получить перечень всех деталей выхлопной системы дизельного двгателя, Вы указываете в запросе на поиск "Д ? Е ???" Пакет программ *Superfile* позволяет осуществить фонетический поиск. Другими словами, Вы можете извлечь из базы данных фамилии, созвучные той, которую Вы ищете, например "Шмит" для "Шмид" и т. д. Такая возможность интересует особенно тех пользователей, которые контактируют с большим количеством клиентов по телефону или непосредственно на рабочем месте.

Говорят, что этот метод, или "алгоритм", как выражаются специа-

а Имя: [Джон] [ ]
б Фамилия: [@ Давенпорт]
в Телефон: [0123*]
г Адрес: [ ]
[Оуксайд]
[ ]
Почтовый индекс: [ ]
Лимит на кредит: [ ]
д Примечания: [* месяц *]
Дата: [> 1.1.84]
Стоимость: [< 1000]
Возврат: [ ]
Баланс: [ ]

Рис. 24. Методы приближенного поиска в системе Superfile:
а – поиск значения "Джон" в первом поле; *б* – поиск значения фамилии, похожей на Давенпорт; *в* – поиск номера телефона, АТС которого имеет номер 0123, * означает, что далее могут следовать любые цифры; *г* – поиск значения "Оуксайд" без указания поля, в котором оно было первоначально записано; *д* – поиск любого клиента, в поле примечаний которого содержится слово "месяц"

листы по вычислительной технике, был изобретен Холлеритом, создателем перфорационных машин, в связи с проведением в 1890 г. в США переписи населения. Бюро по переписи столкнулось с большими трудностями из-за множества людей, эмигрировавших на Золотой берег, и никто не имел ни малейшего понятия о том, как правильно пишутся их неанглийские фамилии. Алгоритм известен под названием *Soundex* [3]. Он работает следующим образом:

1) первая буква слова сохраняется;

2) все буквы, обозначающие гласные, удвоенные согласные и непроизносимые звуки, опускаются;

3) оставшиеся согласные разбиваются на шесть групп и кодируются одним и тем же числом. В каждую группу входят близкие по звучанию звуки, например "д", "т".

Применяя такой алгоритм кодирования для фамилий, мы получим для "Шмид" и "Шмит" одинаковые коды, скажем Ш12, если М = 1 и Т = 2. Таким образом наша цель – найти фамилии, похожие на "Шмит" – оказалась достигнутой.

Алгоритм может успешно применяться для кодирования слов языка, в котором слова пишутся так же, как произносятся, например в испанском языке. С английским языком дело обстоит сложнее. *Tomson* и *Thompson* звучат одинаково и должны быть найдены по алгоритму *Soundex*, но этого не случится из-за буквы *p*. Можно было бы изменить алгоритм и рассматривать *mp* как *m* (как это делается для *ck* и *k*), но как быть со словом *preamplifier*, где *p* произносится.

Помимо символьной информации, в базах данных хранятся цифры. Возможно, Вы захотите узнать, кто должен Вам 43 фунта 56 пенсов, но более вероятно, что Вам понадобится список лиц, задолжавших Вам свыше 1000 фунтов. Это нетрудно сделать, указав в поле, где записывается

кредит, 1000. Но при этом возникает другая важная проблема. Очень часто нам надо хранить целый список значений одного и того же признака. Например, адрес может включать только название организации и город:

адрес 1 Американ экспресс
адрес 2 Брайтон

Но адрес может состоять из нескольких строк:

адрес 1 Даувер Хауз
адрес 2 Мано
адрес 3 Гилдед Лайн
адрес 4 Смарт Крисент
адрес 5 Джувелд Хайтс
адрес 6 Брайтон

Трудность заключается в том, что когда мы ищем информацию о лицах, проживающих в Брайтоне, мы не знаем, искать ли это слово во втором поле или шестом, или еще в каком-нибудь другом.

Американцы и европейцы скажут с самодовольным видом, что это не их трудности. Стандартный адрес в их странах состоит из трех строк.

адрес 1 1140 Винтер Хайтс
адрес 2 Лоуренсвилл
адрес 3 МО 2345

Но не подумайте, что им удастся так легко избавиться от этих проблем. Возьмем, к примеру, список компаний и их директоров. Мы хотим узнать, директором каких компаний является Джонсон:

Компания	"Смол Бизнес"	"Мегагалактика"
Директор 1	Смит	Эндрюс
Директор 2	Джонсон	Шульц
Директор 3		Кризи
Директор 4		Донован
. . . . .		. . . . .
Директор 54		Джонсон

Та же проблема очень остро ощущается в системах документационного обеспечения, которые начали широко внедряться после появления дисков типа "Винчестер" и лазерных дисков большой емкости. В этих системах вместе с документом хранится перечень ключевых слов. Скажем, в медицинских документах, регистрирующих травмы, Вы встретите ключевые слова: "нога", "перелом", "контузия", "интенсивная терапия". Пользователь базы данных может включить в запрос на поиск слова "контузия" и "нога". Он или она не обязаны знать, на каком месте в списке ключевых слов искомого документа находятся эти слова или в каком порядке они следуют. Если пользователь наберет на клавиатуре сначала слово "контузия", а потом "нога", система должна найти документ, несмотря на то, что в списке ключевых слов "нога" встречается раньше, чем "контузия". Возможно, на бумаге проблема выглядит несколько надуманной, но она становится реальностью, когда садишься за терминал ЭВМ.

Фотограф хранит фотографии и их описание. Он использует для опи-

сания фотографии десять ключевых слов. Во время поездки в Шри-Ланка он делает прекрасную фотографию, которую индексирует следующими ключевыми словами: "закат", "сети", "лодка", "рыбак", "Индийский океан" – в такой последовательности. Несколько месяцев спустя ему звонят из журнала и говорят, что нужна фотография рыбака-туземца, возвращающегося вечером домой. Если фотограф не обладает феноменальной памятью (если обладает, то ему база данных не нужна), он вряд ли догадается, что слово "рыбак" было четвертым в списке, а "закат" – первым.

Возможно существуют другие методы решения проблемы, но я о них не слышал. В системе *Superfile* все элементы записи рассматриваются как логически эквивалентные. Вы можете хранить все элементы адреса клиента под единым дескриптором, или ярлыком АДРЕС, и запрашивать затем АДРЕС=ПАСАДЕНА. Вы можете заставить ЭВМ осуществить поиск по команде ДИРЕКТОР=ДЖОНСОН или ТРАВМА=НОГА,или ЯВЛЕНИЕ=ЗАКАТ.

В обычных системах запросы получаются громоздкими, вроде "Если Директор1=Джонсон или Директор2=Джонсон или ... Они не годятся по двум причинам: утомительно набирать такой запрос на клавиатуре: пока Вы не просмотрите базу данных, Вы не узнаете, сколько операторов ИЛИ включать в запрос.

Вы должны уметь объединять условия на поиски в логическое выражение, оно называется булевым (в честь Джорджа Буля – математика, впервые применившего формализованную запись логических выражений). Записанный с помощью булевых выражений запрос на поиск может означать следующее : "Найти всех клиентов, в поле комментариев которых встречается слово "спирт" и задолженность превышает 1000 фунтов, или тех, кто проживает на территории, почтовый индекс которой начинается с букв СВ, и задолженность составляет менее 300 фунтов". Безусловно, поскольку ЭВМ не понимает естественного языка, запрос должен быть записан с помощью символов, входящих в набор символов ЭВМ.

В самом поиске содержится скрытый парадокс. Вы должны знать, что Вы ищете. Это кажется очевидным, однако приводит к запутанным результатам. Существует множество обиходных слов, которые люди каждый раз пишут или сокращают по-разному. Скажем, название Британских графств: "Хертфордшир" или "Хертс"? "Бакингемшир" или "Бакс"? Правильная замена слова "Шропшир" – "Салоп" (краткий вариант латинского названия), но его применяет очень незначительное число людей, остальные пишут "Шропс". В Вашей фирме может работать несколько торговых агентов (Джон, Мери, Гарри, Билл). Их имена включаются в Записи покупателей с тем, чтобы они могли получить причитающиеся им комиссионные. Билл будет крайне раздосадован, если половина его сделок будет связана в базе данных с именем "Вильям", которому и выплатят его деньги.

Выход заключается в том, чтобы проверять информацию при вводе ее в ЭВМ. И если допускается только сокращение "Бакс", то СУБД не примет значения "Бакингемшир" и Вам придется разобраться, чего же Вы все-

таки хотите. Хороший пакет программ управления базой данных должен иметь в составе программу, которая сравнивает слова, вводимые пользователем, с перечнем разрешенных в системе слов. Разработчики должны продумать все эти вопросы, прежде чем начинать проектировать базу данных.

3.5. УДАЛЕНИЕ ЗАПИСЕЙ

Основная функция системы управления базой данных очень проста. Она заключается в хранении и изменении записей. Самая сложная из перечисленных операций – внесение изменений. Преимуществом записей постоянной длины является то, что новая запись имеет такую же длину, какую имеет старая, и может быть записана на ее место. В системе с записями переменной длины необходимо сначала удалить старую запись, затем внести изменение и записать снова. Беда в том, что люди (а их не переделаешь) очень часто удаляют информацию, без которой вскоре не могут обойтись. В системе *Superfile* мы пытались справиться с этим человеческим недостатком следующим образом – запись удаляется в два этапа. Сначала она лишь помечается как удаленная и физически не меняется. При необходимости ее можно извлечь с помощью специальных команд. На втором этапе происходит "чистка" базы данных от всех записей, помеченных как удаленные. На этот раз они уничтожаются физически, а оставшиеся записи сдвигаются, чтобы использовать пространство в памяти (эти же задачи, только более сложные, решает операционная система, см. гл. 1). Само собой разумеется, что процесс перемещения записей делает бесполезным существующие индексы, которые должны создаваться заново.

3.6. ГЕНЕРАТОР ОТЧЕТОВ

После того как записи введены в базу данных, проверены, изменены и некоторые удалены, их можно просмотреть одну за другой на экране, используя специальную программу. В том случае, когда пользователю нужна масса записей – чтобы подвести итоги продажи, написать типовое письмо всем задолжникам или напечатать библиографию по какому-либо вопросу – он использует генератор отчетов.

Генератор отчетов работает так же, как и генератор форм в том смысле, что он извлекает записи из базы данных в соответствии с поступившим запросом и представляет их в заданном формате, только не на экране терминала, а на печатающем устройстве ЭВМ, где записи распечатываются одна за другой. Поскольку отчет представляет собой таблицу, в названии генераторов отчетов часто можно встретить слово ТАБ, например EILETAB, SUPERTAB.

Проектировать форму представления отчета, так же как и форму для ввода данных, должен специалист, немного разбирающийся как в вычислительной технике, так и в деятельности предприятия. Использовать отчет может совершенно другое лицо, ничего не знающее и не желающее знать что-либо о компьютерах.

Люди имеют различное представление об отчетах, и я высказываю сугубо свою точку зрения о структуре отчета.

а	Фирма АВС Товарооборот за январь 1983 г.	
б	Торговый посредник	Объем продажи
в	Эндрюс	450.23
	Браун	1292.50
	Лайтмен	65.92
	Продано всего по Берксу	1808.65
	Число посредников: 3, среднее	602.88
г	Чандлер	650.76
	Джонсон	156.89
	Трант	2987.00
	Уильямс	56.72
	Продано всего по Баксу	3851.37
	Число посредников: 4, среднее	962.84
	Аккерман	633.70
	...	
	...	
д	Общий объем продажи	36293.69
	Число посредников: 47, среднее	772.20

Рис. 25. Структура отчета в системе Superfile включает пять основных групп. Содержание отчета представлено в виде таблицы, элементы которой заполняются из базы данных. Они могут быть отсортированы, просуммированы. С ними можно выполнять другие вычисления:

а – заголовок отчета, напечатанный на первой странице; *б* – заголовок страницы (программный модуль, формирующий эту часть отчета, включает также счетчик страниц); *в* – строки отчета, содержащие фамилии торговых представителей и значения объемов продажи (фамилии отсортированы в алфавитном порядке); *г* – промежуточные итоги, выводятся на печать после того, как меняется значение управляющего поля Графство (в данном случае с Беркс на Бакс); *д* – строка, содержащая общий итог отчета

Отчет состоит из пяти элементов:

1) заголовок отчета размещается на первой странице и объясняет назначение документа. Если отчет подготавливается периодически, то пользователь вводит в ЭВМ значение нескольких полей: дату составления отчета, фамилию ответственного за подготовку отчета и т. д.;

2) заголовок страницы объясняет наименования столбцов;

3) строки отчета составляют основу его содержания. Они состоят из заранее определенных полей, значения которых извлекаются системой из базы данных. В примере на рис. 25 отчет содержит только одну строку, если бы мы печатали фамилии и адреса, могло быть пять или шесть строк. Программа "генератор отчетов" должна выполнять расчеты внутри строки – так же как это делает программа, предназначенная для вывода форм;

4) итоговые строки. Если отчет предварительно был отсортирован по определенным ключам, например в порядке алфавита американских штатов, то данные об объемах продаж торговых посредников в пределах од-

ного штата могут быть суммированы автоматически, как только название штата изменится с "Канзас" на "Миссисипи". Программа должна подсчитывать промежуточные итоги по нескольким столбцам в строке, а также подсчитывать число слагаемых и определять средние значения и т. п.;

5) наконец, отчет может заканчиваться общим итогом, который складывается из всех промежуточных итогов, полученных в отчете. Генератор отчетов может выполнять еще одну полезную функцию – создавать на экране форму и выводить информацию, содержащуюся в отчете, на экран. В этом случае пользователь может задать условия поиска и снять твердую копию только для отчета, в котором фигурируют торговые посредники с объемом продажи более 10 000 фунтов стерлингов.

3.7. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ БАЗЫ ДАННЫХ

Программы, входящие в ядро СУБД, состоят из двух частей: одна занимается тем, что безропотно принимает на хранение и выдает необходимую информацию, вторая позволяет программировать на некотором языке различные задачи с использованием этой информации, другими словами, управляет работой первой части программного пакета.

В настоящее время самым распространенным программным обеспечением баз данных для микроЭВМ является пакет *dBaseII*. Он выполняет невидимые для пользователя операции индексирования, хранения и поиска записей и имеет в своем составе язык программирования Д, на котором пользователь пишет прикладные программы для решения практических задач. Безусловно, применяемые СУБД для микроЭВМ – это лишь сильно упрощенные варианты СУБД для больших ЭВМ, которые развивались долгие годы. По ним написаны толстенные тома специальной литературы, разработаны очень дорогие программные комплексы, выполняющие разнообразные функции. Другая проблема, связанная с традиционными базами данных, заключается в том, что они требуют сверхпрофессионализма. Вплоть до недавнего времени базами данных управляли опытные специалисты по вычислительной технике, которые окружали свою работу ореолом таинственности с тем, чтобы упрочить свое положение в организации. Действительно, в некоторых организациях, например в коммерческих банках, база данных о клиентах может быть источником дохода и статус специалистов по вычислительной технике там очень высок. В условиях отсутствия конкуренции такие профессиональные группы неизбежно сбиваются на путь решения чисто технических проблем и большое число работ по базам данных посвящено изложению математических принципов, реализация которых открывает перед программистами широкие возможности показать свое искусство. Недавно специалисты фирмы "Дейта электроник корпорейшн" выпустили обзор по базам данных, где серьезно критиковали программистов за отступление от нормальной формы и поддержку повторяющихся групп данных. Повторяющаяся группа данных – это очень полезное свойство программ баз данных, которое мы разбирали

в гл. 2 на примере поля "Адрес", и ради преимуществ, которые несут с собой повторяющиеся группы данных, возможно стоило бы отказаться от нормальной формы. Этот факт показывает, насколько профессиональные разработчики баз данных витают в заоблачных высотах теории, оставляя без внимания полезные для заказчика новшества.

Основная сложность разработки общих пакетов для управления базами данных состоит в том, что требования к базам данных со стороны заказчика непредсказуемы. Все, кто садятся за микроЭВМ и работают с пакетами программ обработки текста, имеют дело с одними и теми же объектами: буквами, словами, параграфами, главами. Создатели программ обработки текстов учитывают опыт редакторской работы, накопленный человечеством за два тысячелетия, и могут предугадать требования заказчика. Требования к программе расчета рабочих таблиц также очень легко предугадать, хотя сами программы написать не так просто. Автор использовал хорошую идею, основанную на пятисотлетней практике бухгалтерского учета. К сожалению, каждая организация разрабатывает собственную базу данных. Для одного пользователя важно хранить список покупателей и их заказы, другому нужны координаты движения руки робота и сигналы для шести шаговых электродвигателей, осуществляющих ее перемещение. Третий пользователь хочет иметь базу данных о Гималайских хребтах, горных перевалах и юридических формальностях, связанных с их прохождением. Четвертый хранит данные о снятых фильмах, специалистах, работающих в области кино и хотел бы узнать, кто снял три или более фильмов, принесших доход более 50 млн. фунтов стерлингов. Пятый записывает в базу данных важнейшие политические события в мире и его интересует перечень всех событий на Ближнем Востоке, в описаниях которых встречаются слова "нефть" и "взрывы", а также события, предшествующие этим событиям.

Некоторые проблемы, возникающие при проектировании баз данных, являются общими. Однако чаще они вызваны спецификой баз данных. Все это ставит разработчика СУБД в довольно сложное положение – он может предусмотреть только часть требований. В конце концов,не лучше ли предоставить пользователю возможность писать самому прикладные программы для решения своих задач? Мы отложим обсуждение этой проблемы до гл. 5.

3.8. БАЗЫ ДАННЫХ ДЛЯ ХРАНЕНИЯ ТЕКСТОВ

С появлением микроЭВМ люди находят все новые и новые задачи, о решении которых на больших машинах раньше никто и не помышлял. Одна из них – создание баз данных для хранения текста. Такие базы данных нужны во многих областях деятельности. Встречаются базы данных, где символьная информация хранится наряду с цифрами: в медицинских отчетах, исследованиях, лабораторных анализах и т. д. Интересным примером служит библиографическая база данных. Каждая запись в такой базе обычно состоит из следующих элементов: порядковый номер,

Перед Вами пример обработки текста. Применение ЭВМ существенно снижает трудоемкость машинописных работ. Оператор может изменить формат и структуру документа, исправить опечатки.
Эта книга написана о компьютерах, а компьютеры сами по себе без програ|м| ничего не значат. Програ|м|а определяет функции машины. Быстрое распространение компьютеров означает все увеличивающийся спрос на програ|м|истов.

В первом варианте этот параграф следовал за параграфом, в тексте которого слово "программа" и его производные были ошибочно написаны с одной буквой "м".

Перед Вами пример обработки текста. Применение ЭВМ существенно снижает трудоемкость машинописных работ. Оператор может изменить формат и структуру документа, исправить опечатки.

В первом варианте этот параграф следовал за параграфом, в тексте которого слово "программа" и его производные были ошибочно написаны с одной буквой "м".

Эта книга написана о компьютерах, а компьютеры сами по себе без программ ничего не значат. Программа определяет функции машины. Быстрое распространение компьютеров означает все увеличивающийся спрос на программистов.

Рис. 26. Пакет программ для обработки текста MINCE позволяет уменьшить формат, переставить параграфы, исправить слова или опечатки в них. Текст можно сохранить на диске и снова напечатать. При этом экономится огромный труд по перепечатке вариантов. Потребность в базах данных, которые могут хранить текст, все время возрастает

заглавие, автор(ы), издательство или источник и дата, число страниц, число библиографических ссылок, краткое изложение содержания. Длина каждого поля постоянно меняется и хранить такую запись в базе данных с записями постоянной длины трудно, а из-за последнего элемента – краткого изложения содержания, которое может занимать от 0 до 1000 строк – вообще невозможно. Ввод и редактирование такой порции текста требует применения программ обработки текстов совместно с базой данных с переменной длиной записи. В противном случае использование па-

мяти ЭВМ будет неэффективным. Для такого приложения требуется соединить возможности базы данных и редактора текста. Поскольку другие решения мне неизвестны, я постараюсь объяснить, как мы удовлетворили этому требованию при создании пакета *Superfile*. Мы взяли прекрасный пакет для редактирования текста *MINCE**, созданный фирмой "Марк оф Юникорн" (другие названия этого пакета *The Final Word* и *Perfect-writer*). Он стал отличной основой того редактора текста, который мы встроили в *Superfile*.

MINCE предусматривал работу с двумя окнами на экране и возможность одновременного редактирования до семи документов. Их содержание отражалось на экране попарно. Все, что нам нужно было сделать — это объединить *MINCE* с существующей программой вывода формы на экран таким образом, чтобы пользователь мог перемещать курсор, как это обычно делается при редактировании текста, но только в пределах полей, предусмотренных формой.

Для того чтобы обрабатывать элементы информации переменной длины, мы сделали поля "подвижными". Их размер начинается с одного символа и постепенно увеличивается по мере ввода текста. Во многих случаях заполненные поля не отличались от полей постоянной длины, так как значения данных имели стандартный размер. Однако поля, содержащие текст (рис. 26), могли быть расширены до размеров текста.

Повторяющиеся группы данных позволяют иметь списки признаков в записи. Эта возможность оказалась настолько полезной, что мы решили сохранить ее в своем варианте редактора текста. Специальный символ, которым заканчивается поле, показывал программе, что необходимо создать новое поле с тем же именем.

На пути создания эффективной базы данных для хранения текста важно было решить еще две проблемы. Конечная цель разработки базы данных состоит в том, чтобы находить записи быстро, организуя поиск по ключу внутри записи. Просмотр значений небольших по размеру полей не представляет трудности, но как быть с большими текстовыми полями? Это можно сделать, применив обычный поиск "по образцу" внутри большого неизвестного куска текста**. Однако такой поиск займет слишком много времени. Система должна просматривать каждое слово текста, хранящегося в соответствующем поле. Для ускорения поиска пользователю предоставляется возможность выделить в тексте с помощью курсора ключевые слова и объединить их в набор ключевых слов, нажав на управляющую клавишу.

---

* *MINCE* основывается на редакторе текстов *EMACS*. Этот редактор создан для большой ЭВМ в Массачусетском технологическом институте. Слово *MINCE* представляет собой сокращение от *MINCE Is Not Complete EMACS* — неудачный пример страсти программистов к рекурсии.

** Если Вы хотите найти текст, содержащий в середине слово "комары", например "С наступление темноты комары стали невыносимы", то в задании на поиск Вы должны указать "* комары *". Звездочка означает любой текст.

Наконец, нам пришлось усовершенствовать программу сортировки с тем, чтобы в базе данных хранились отсортированные записи по каждому элементу, входящему в повторяющуюся группу данных. Например, если информация о книге, написанной тремя авторами Брауном, Смитом и Джонсом, хранится в библиографической базе данных, то логично при распечатке получить три записи, расположенные в порядке алфавита авторов:

Браун, Смит, Джонс; Большие горные хребты
Джонс, Браун, Смит; Большие горные хребты
Смит, Браун, Джонс; Большие горные хребты.

3.9. БАЗЫ ДАННЫХ ДЛЯ ХРАНЕНИЯ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

Современные персональные микроЭВМ имеют экраны с высокой разрешающей способностью, а их печатающие устройства могут воспроизводить довольно сложные графические изображения. Не существует технических трудностей для того, чтобы объединить базы данных, в которых хранятся изображение и текст.

Изображение хранится в базе данных в виде цепочки символов и выводится на экран в отведенное для этих целей поле. Генератор форм распознает это поле по специальному коду и "рисует" в нем.

Картинка на экране воспроизводится довольно простым способом. Экран разбивается на множество небольших участков, которые могут быть белыми или черными или промежуточного оттенка в зависимости от значения байта в памяти ЭВМ, связанного с конкретным участком. Участки играют роль элементов изображения. Если изображение цветное, то элемент окрашен в один из трех основных цветов. Изображение составляется из большого числа элементов, считываемых из файла (в нашем случае из файла базы данных). Пока, насколько мне известно, никто подобного пакета программ не разработал, но принципиальных трудностей для создания подобного программного обеспечения нет, нужны только соответствующие технические средства.

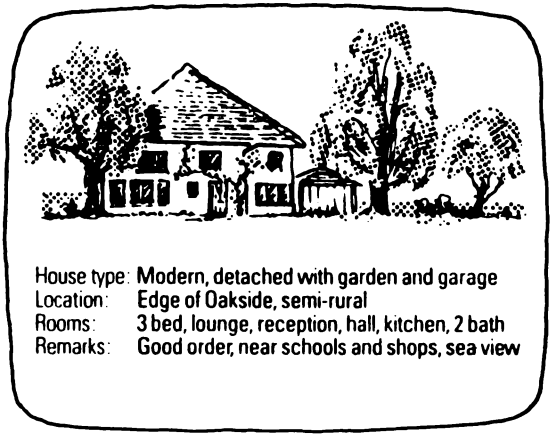


Рис. 27. С развитием технических средств появляется возможность хранить в базе данных не только текст, но и изображения. На рисунке воспроизведена запись базы данных о продаже недвижимого имущества

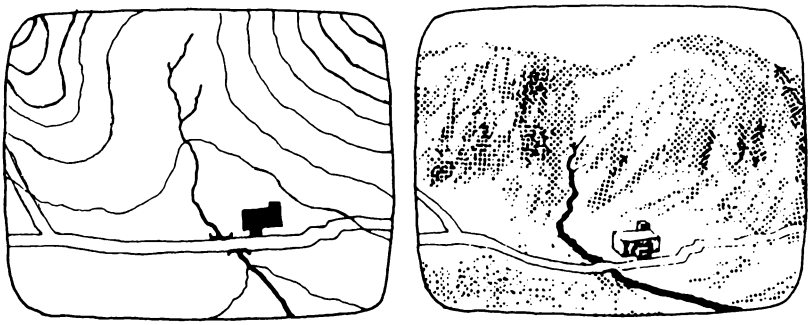


Рис. 28. Карта может храниться в базе данных и отображаться на экране дисплея либо обычным образом (слева), либо в виде картинки, наблюдаемой пилотом самолета (справа)

Запись базы данных, содержащая графическое изображение и текст, могла бы выглядеть на экране не так, как это показано на рис. 27.

Такого рода изображение не имеет какой-либо структуры и представляет собой простую совокупность байтов. Интереснее, когда картинка может составляться из отдельных элементов. Например, в базе данных уголовного розыска хранятся элементы изображения, из которых составляется фоторобот. Свидетели могут показать, что разыскиваемый преступник имеет узкий разрез глаз, лысую голову, широкие скулы, усы и т. д. Соответствующие элементы изображения, собранные вместе СУБД, образуют фоторобот, при этом программное обеспечение базы данных создает также словесное описание внешности преступника. По этому описанию система может найти записи преступников, черты внешности которых классифицируются аналогичным образом. Зубной врач может хранить в такой базе данных записи о своих пациентах. Эти записи содержат сведения в удобной форме о состоянии зубов, об элементах зубных протезов. Кроме того, если врач поставил пломбу, система помещает специальный код в соответствующее поле и хранит словесное описание в записи базы данных.

Важным приложением является электронная картография. Если Вы откроете географический атлас, то увидите, что в начале атласа помещены карты, а в конце – перечень географических названий с указанием их широты и долготы. Описание местности, хранящееся в базе данных, позволяет найти с помощью несложной программы оригиналы самих карт и отобразить их на экране. Добавьте к ним сведения о населении, реках, холмах, железных и шоссейных дорогах и т. д. и Вы получите базу данных, которая очень заинтересует работников сферы обслуживания.

Географическая карта представляет собой уменьшенное и обобщенное изображение участка земной поверхности, полученное с далекого расстояния. Можно этот же участок отобразить на экране дисплея в виде картинки, наблюдаемой с более близкого расстояния, например из иллюминатора

самолета, летящего на высоте 100 м в северном направлении. Специалисты военно-воздушных сил США, занятые разработкой самых сложных технических систем, в настоящее время работают над техническими средствами ЭВМ с тем, чтобы отобразить летные карты именно таким образом. После того как карта введена в базу данных, она может корректироваться в реальном масштабе времени, т. е. во время проведения аэрофотосъемок.

3.10. ПРОГРАММЫ РАСЧЕТА РАБОЧИХ ТАБЛИЦ

Большую пользу может принести также совместное использование пакетов программ расчета рабочих таблиц и управления базой данных. С точки зрения тех, кому не пришлось преодолевать трудности программирования рабочих таблиц, этот пакет программ представляет собой электронный вариант бухгалтерского отчета.

Можно сказать, что именно этот пакет, известный под названием *Visicalc*, открыл широкую дорогу микроЭВМ в сферу конторского труда. Пакет программ *Visicalc* принес его создателям большой успех. Благодаря тому, что фирма "Эппл" включала в состав программного обеспечения своих микроЭВМ этот пакет, ей удалось в начале 80-х годов продать рекордное количество персональных компьютеров. Электронная рабочая таблица состоит из строк и колонок цифр (можно вводить и текст, обычно он располагается в головке и боковике таблицы) и пунктирных линий, с помощью которых в таблице выделяются итоги. Значения элементов таблицы могут быть заданы пользователем с клавиатуры или рассчитаны по формулам, введенным пользователем. Так, поступление денежных

Фирма "Джо Соуп"				
Кварталы	1	2	3	4
РАСХОДЫ				
Численность	9	10	11	12
Зарплата	63000	68600	74760	81536
Накладные расходы	36000	39200	42720	46592
Материалы	40000	44000	48400	53240
Прочие расходы	8000	8800	9680	10648
Всего расходов	147000	160600	175560	192016
Продано изделий	8000	8800	9680	10648
ДОХОДЫ				
Объем продажи	152000	167200	183920	202312
Квартальные отчисления	5000	6600	8360	10296
Баланс в банке	– 20000	– 15000	– 8400	– 40

Рис. 29. Пакет программ расчета рабочих таблиц является вероятно наиболее популярным пакетом готовых программ для микроЭВМ. Его можно рассматривать как небольшую СУБД, в которой строки играют роль информационных признаков, а столбцы – записей

средств в кассу предприятия, изготавливающего специальные гаечные ключи, может быть рассчитано, если известна недельная выработка рабочих (например, трое рабочих могут изготовить за неделю 100 ключей). Заработная плата и накладные расходы подсчитываются автоматически. Объем реализации определяется по цене, установленной на ключи. Прибыль предприятия равна объему реализации минус затраты на производство. Задав эти зависимости, управляющий сможет смоделировать размеры прибыли для различных уровней производительности труда.

При использовании обычного пакета программ расчета рабочих таблиц предполагается, что все исходные данные вводятся пользователем. Однако, автоматизированные системы управления предприятиями почти наверняка имеют эти данные в базе данных. Чтобы увидеть их на экране в форме таблицы, необходимо извлечь информацию из базы данных. Поэтому пакет программ расчета рабочих таблиц служит своего рода интерфейсом между конечным пользователем и базой данных.

Форма представления информации, принятая в электронной рабочей таблице, не является необычной: если представить себе базу данных в виде строк (признаки) и столбцов (записи), то мы немедленно получаем рабочую таблицу. В любой момент времени Вы можете просмотреть часть базы вместе с итогами, показывающими эффективность предыдущих вариантов.

		Записи			
		Январь	Февраль	Март	Апрель
Признаки	Зарплата	2300	2300	2400	2400
	Накладные расходы	1600	1600	1700	1700
	Реклама	2000	2000	1800	1800

В жизни все значительно сложнее, поскольку реальные записи будут более детальными, чем показано здесь. "Зарплата в феврале" (2300) представляет собой сумму многочисленных выплат отдельным работникам в течение этого месяца. Накладные расходы получаются путем суммирования многих статей расходов: плата за помещение, электро- и теплоснабжение и т. д. Информация о каждой из хозяйственных операций хранится в виде записи в базе данных. Поэтому извлечение информации из базы данных и представление ее в форме рабочей таблицы может оказаться совсем не простым делом, как это кажется на первый взгляд. Более того, искомые записи могут оказаться "виртуальными", т. е. составленными из других записей, распределенных по реляционной базе данных (см. гл. 4).

Пока я не встречал пакетов программ, которые успешно бы выполняли эти функции. Существуют пакеты, например *Lotus* 123, *The incredible Jack*, *T=Maker*, которые реализуют часть функций, используя редактор текстов, но, на мой взгляд, они работают неудовлетворительно. Пакеты не являются настоящими СУБД в том смысле, что они не могут осуществить

быстрый поиск в больших базах и составить виртуальную запись из записей различных типов.

Разработчикам программного обеспечения придется подождать, пока мы не научимся более точно определять типичные операции, необходимые конечному пользователю. Только тогда удастся написать хорошие пакеты программ. Еще раз хочу подчеркнуть, что нетрудно написать такую программу для единственного заказчика при условии, что он располагает достаточными средствами, чтобы оплатить эту работу. Трудно написать программу, которую могло бы настроить на свои практические задачи множество рядовых пользователей.

3.11. МАШИННАЯ ГРАФИКА

Существует мнение, что цифры, этот "исходный материал" для любой хозяйственной операции, не воспринимается руководителем так наглядно, как графики, особенно если они цветные. Об объемных диаграммах и говорить не приходится – они приводят руководителей в восторг. Я подозреваю, что за этими утверждениями скрывается страстное желание фирм-производителей как-то выделить свои микроЭВМ из числа других, аналогичных по функциям машин.

Во время проведения выставок вычислительной техники бросается в глаза восторженная группа посетителей, толпящаяся вокруг терминала, на котором вычерчивается цветное изображение. Неважно, что вычерчивается, важно, что изображение цветное. Яркое, сочное и бессмысленное, как рисунки в детском саду. Чаще всего ЭВМ рисует изометрический график какой-либо несложной математической функции. Физикам они нужны, чтобы представить различные математические идеи, но для хозяйственной деятельности они бесполезны, и вряд ли найдется хотя бы один бизнесмен, который понимает, на что он смотрит. Однако все это выглядит впечатляюще и технически очень сложным. Поэтому громадные усилия затрачиваются на то, чтобы заставить ЭВМ рисовать на экранах и печатающих устройствах столбиковые и круговые диаграммы. Помимо технических средств требуются довольно трудоемкие программы, чтобы отобрать таблицы цифр из базы данных и превратить их в графики или диаграммы.

Месяц	Январь	Февраль	Март	Апрель	Май	Июнь
Продано	12	14	23	26	18	10

Независимо от того, извлекаете ли Вы эти цифры из базы данных или из программы расчета рабочих таблиц, проблемы возникают общие. Итоги продажи за месяц состоят из многих отдельных хозяйственных операций, которые надо суммировать с помощью отдельной программы, например, генератора отчетов. Программа должна выполняться несколько раз, чтобы получить итоги за каждый месяц. Прежде, чем построить график, программа просматривает все итоговые значения, выбирает наибольшее и

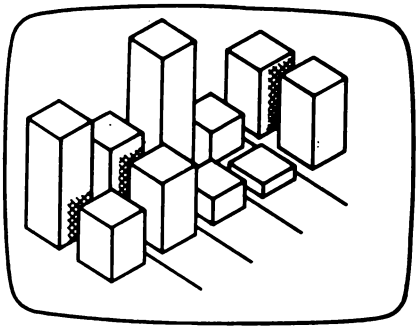


Рис. 30. Покупатели микроЭВМ требуют, чтобы машины умели рисовать столбиковые диаграммы. Но есть ли от таких диаграмм какая-нибудь польза?

наименьшее значения, с их помощью рассчитывает масштаб, что позволяет правильно разместить график на странице. После этого программа строит график. Трудно представить себе универсальную программу, которая могла бы решать самые различные задачи средствами машинной графики. Иногда руководителю кажется, что круговая диаграмма выглядит лучше. В этом случае программа должна сложить исходные данные, определить их долю и разделить 360° на пропорциональные части. К недостаткам графиков относится и то, что форма одной и той же кривой может меняться самым различным образом в зависимости от выбранных масштабов и начала координат. Так, на рис. 31, *а*, где значение 10 принято за начало отсчета, перепады графика выглядят очень тревожными, а на рис. 31, *б*, где за точку отсчета принят нуль, этот график менее впечатляет. Если использовать логарифмическую шкалу по оси *y* рис. 31, *в*, то график становится еще более плоским.

Такие приемы хорошо известны всем, кто занимается построением графиков, и вряд ли целесообразно применять их автоматически.

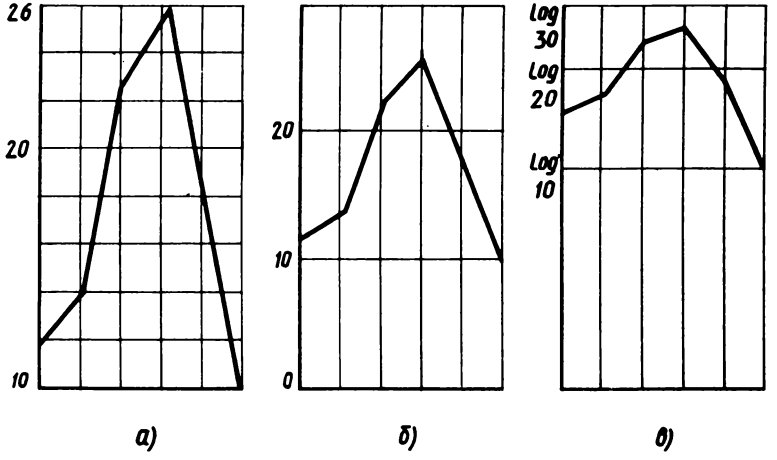


Рис. 31

ГЛАВА 4
МНОГОФАЙЛОВЫЕ БАЗЫ ДАННЫХ

До сих пор мы рассматривали базу данных, в которой все Записи были одного типа. Такое допущение позволило упростить изложение материала. В реальных системах почти наверняка будут храниться взаимосвязанные Записи различных типов, которые в зависимости от принятой терминологии образуют несколько баз данных или несколько "файлов" (см. § 1.2, в котором речь идет о файлах ЭВМ). Часто используются и другие термины: "логическая база данных", "плоский файл" или "логический файл". Необходимость различать типы Записей существует и в системе *Superfile*, где информация физически хранится в одном единственном файле. Например, фамилия и адрес клиента – это Запись одного типа, сведения о любой хозяйственной операции – Запись другого типа, данные о проданном товаре – Запись третьего типа и данные о фирме-изготовителе – Запись четвертого типа.

В библиографической базе данных название книги, фамилия автора, название издательства, номер книжной полки и т. д. могут составлять одну Запись, фамилия абонента и дата обращения за книгой – другую. В базе данных о деятельности компании Запись одного типа содержит информацию об организационной структуре компании, Запись другого типа – о ее ежегодных доходах.

Авиакомпания может иметь базу данных, в которой хранятся Записи о технических характеристиках самолетов, кадровом составе (фамилии работников, их адреса, квалификация и др.), о предоставляемых пассажирам услугах, о выполняемых рейсах (тип самолета, состав экипажа, местонахождение лайнера, список пассажиров).

Во всех приведенных примерах Записи различных типов так или иначе связаны друг с другом. Это не всегда так. В базе данных, в которой хранится политическая или экономическая информация, Запись о каждом событии должна быть связана с предыдущими Записями того же типа. Например, нефтяная компания объявляет о начале промышленной эксплуатации нефтяного месторождения в районе, ранее считавшимся бедным на природные ископаемые. Несколько месяцев спустя местное правительство открывает там авиалинию, а годом позже ряд коммерческих банков Лондона и Нью-Йорка открывают свои филиалы в столице этого государства. Запись о событии вызывает каскад новых Записей.

В состав Записи входят различные элементы информации. В теории баз данных одним из самых неразработанных вопросов остается вопрос о том, какие элементы информации включать в Запись одного типа, а какие в Запись другого типа. Состав Записей определяется здравым смыслом человека, который отвечает за разработку и внедрение системы. Обычно объединяют данные, значения которых меняются нечасто. Например, люди проживают длительное время по одному и тому же адресу. Для многих баз данных оказывается целесообразным хранить в одной Записи фамилии и адреса. Однако, если в базе хранятся сведения о Ваших торго-

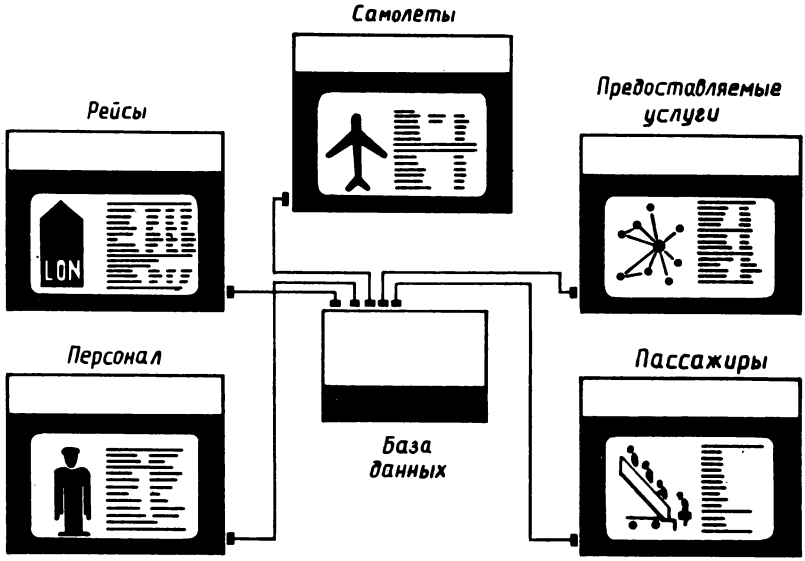


Рис. 32. Авиакомпания использует несколько баз данных, содержащих информацию о расписании рейсов, обслуживании пассажиров, эксплуатации самолетов и т. д.

вых представителях, которые часто разъезжают по командировкам, то лучше хранить фамилию представителя и домашний адрес в одной Записи, а адрес гостиницы, где он или она проживает в данный момент — в другой. Более того, возможно, что гостиниц, где когда-либо останавливались Ваши торговые представители, наберется всего несколько сотен. В этом случае удобно весь список гостиниц хранить постоянно в базе данных, разработав при этом некий алгоритм, позволяющий определить, в какой гостинице какой представитель остановился на определенную дату. Именно наличие такого алгоритма делает работу с базой данных интересной.

Самым простым решением проблемы распределения элементов информации по Записям различных типов было бы включение всех элементов информации в каждую Запись. В этом случае база данных авиакомпании содержала бы огромные по объему Записи, куда входили бы сведения о самолетах, их технические характеристики, имена и адреса членов экипажа, фамилии всех пассажиров, описание груза и т. д. Всякий раз, когда назначался бы новый рейс, всю эту информацию нужно было бы переписывать в новую Запись.

В базе данных отдела материально-технического снабжения предприятия по каждой накладной надо фиксировать фамилию и адрес клиента, дату, покупки, торговую фирму, остаток данного материала на складе и т. д. Такой объем информации был бы излишним для решения конкрет-

ных практических задач, например подготовки с помощью ЭВМ документа на транспортирование груза. Подобный подход означает многократное дублирование информации. Скажем, в каждой Записи о хозяйственной операции указываются полностью фамилия клиента, его адрес, сведения о платежеспособности и т. д. Такая организация базы данных очень несовершенна, сильно замедляет поиск и нерационально использует дорогую дисковую память. Но хуже всего то, что такая база данных функционально просто не работает. Если наш клиент изменил адрес, мы должны просмотреть всю базу данных и изменить все Записи. Предположим, мы хотим сохранить старый адрес. Тогда мы вынуждены сохранить описание одной или нескольких хозяйственных операций, в которых упоминается старый адрес. Один из основных принципов, на которых строится база данных, заключается в том, что каждый элемент информации вводится в базу один раз и к нему можно получить быстрый доступ.

К счастью, многие реальные электронные хранилища информации состоят из Записей единственного типа. Но имеется также немало баз данных, в которых невозможно ограничиться одним типом Записей. Лучший выход из положения в этом случае – хранить в базе данных множество небольших Записей, которые автоматически связываются друг с другом. Как осуществить такую связь? Очевидное решение часто оказывается и самым лучшим. Состоит оно в том, что две Записи, которые Вы хотите связать друг с другом, содержат общий элемент данных, который нигде больше не повторяется. Поскольку в базе данных все элементы индексируются, то каждая Запись "указывает" на другую, каждая Запись может быть найдена вслед за другой Записью. Безусловно, каждая из этих Записей может включать элементы, которые указывают на одну или более Записей других типов. Рассмотрим подробнее, как реализуется такой подход к базе данных предприятия.

В системе *Superfile* каждая Запись состоит из наименований полей и их значений. Например, по каждому покупателю имеется такая Запись:

Имя	= Мери
Фамилия	= Смит
Адрес	= Ларчис
Адрес	= Магнолия стрит
Адрес	= Кингстон
Идентификационный номер	= 34924

Всякий раз, когда мисс Смит покупает что-либо, система формирует следующую Запись учета продаж (идентификационный номер выступает в качестве элемента – связки):

Идентификационный номер	=34924
Дата	=24 февраля 1984
Буквенный код	=асс
Инвентарный номер	=66831

Эта Запись менее понятна, но из нее следует, что 24 февраля 1984 г. клиент с идентификационным номером 34924 приобрел изделие, буквен-

ный код которого аас (этот код может означать складское помещение), а инвентарный номер 66831.

Остается неясным, что же все-таки было куплено? Что означает "аас 66831"? Запись учета товарно-материальных ценностей дает ответ на эти вопросы:

Буквенный код	= аас
Инвентарный номер	= 66831
Описание	= Электрическая пишущая машинка "Гипертроник"
Изготовитель	= фирма "Гипоид тайпрайтер"
Тип изделия	= 34251
Серийный номер	=229221
Цена	= 221.67

Эта Запись говорит о том, что интересующее нас изделие представляет собой электрическую пишущую машинку марки "Гипертроник", изготовленную фирмой "Гипоид тайпрайтер". Мы знаем теперь тип изделия и его серийный номер, и, что самое важное, сколько оно стоит.

Запись следующего типа содержит информацию о нашем заказе предприятию оптовой торговли на поставку дюжины таких машинок, одна из которых уже доставлена:

Дата	= 1 ноября 1983
Инвентарный номер	= 66831
Номер поставщика	= 4545
Стоимость	= 1862,03
Количество	= 12

И, наконец, имеется Запись о поставщике:

Компания	= "Стеллар стейшенери хоулсэйл"
Адрес	= П/О Бокс 2
Адрес	= Нью-Йорк
Номер поставщика	= 4545

Реляционная база данных, содержащая Записи этих четырех типов, схематично может быть представлена так, как это показано на рис. 33.

Мы можем ответить теперь почти на любой вопрос, связанный с деятельностью нашего предприятия, можем объединить родственные элементы Записей и составить документ – накладную или выбрать соответствующие показатели и подготовить финансовый отчет. Если вдруг выяснится, что конструкция механизма возврата каретки для машинок с серийными номерами между 229200 и 229300 имеет скрытый дефект, мы можем без труда найти в базе данных, кто купил эти машинки и уведомить об этом покупателей. Допустим, один из наших специалистов отправляется в командировку в г. Кингстон. Мы можем выбрать из базы данных фамилии всех покупателей, которых он должен предупредить о дефекте по телефону, находясь в Кингстоне. Как осуществляется такой поиск?

Нам известно только, что Описание включает слово "Гипертроник",

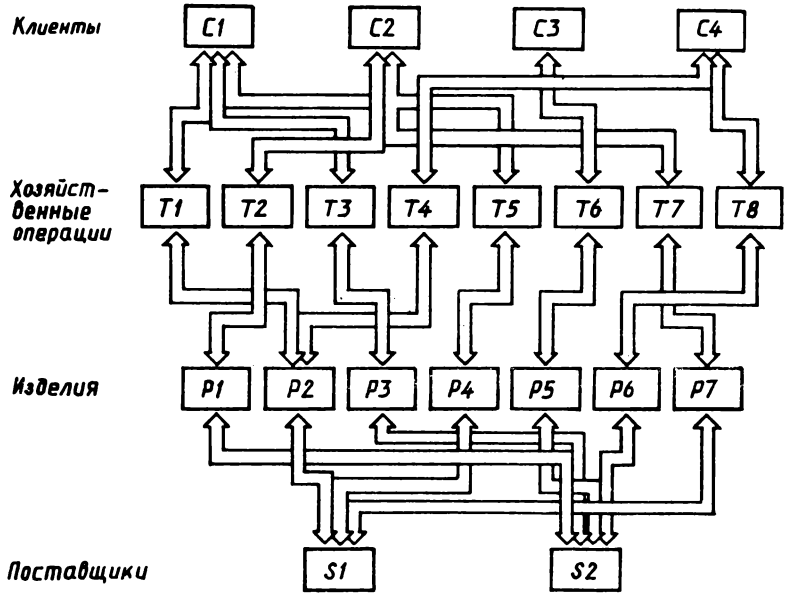


Рис. 33. В реляционной базе данных записи указывают друг на друга, поэтому информация вводится только один раз. На рисунке представлена база данных, содержащая сведения о покупателях, хозяйственных операциях, изделиях и поставщиках

а Серийный номер больше 229200 и меньше 229300. Поиск по этим признакам позволяет извлечь из базы данных запись учета товарно-материальных ценностей, Буквенный код в которой равен "аас", а Инвентарный номер равен 66831.

Используя эти значения в качестве ключей поиска, мы можем найти все Записи учета продаж, в которых говорится о покупке пишущих машинок этой марки и среди них Запись, содержащую идентификационный номер 34924. Теперь уже поиск ведется по условию: Идент. номер = 34924 и Адрес = Кингстон. В результате среди прочих мы находим мисс Смит.

В обычных базах данных типы записей вполне определенные. Записи о покупателях имеют один фиксированный формат, Записи учета продаж – другой. Заранее известно, какие поля входят в состав Записи того или иного типа. В программы, обеспечивающие реляционный поиск, достаточно ввести данные о том, какой тип Записи следует искать, и что искать внутри Записей этого типа. В системе *Superfile* нет записей фиксированной длины: любая Запись может иметь число полей, отличное от предыдущей.

Каждый раз, осуществляя поиск по признакам, извлеченным из предыдущей Записи, мы находим следующую запись по наименованию поля, которое встречается только в этой Записи. Например, если ввести в базу

данных запрос на поиск Записей, содержащих идентификационный номер = 34924, мы найдем не только Запись о покупательнице Мери Смит, но и Записи о всех ее покупках. В данном случае нас интересует Запись о покупателе. Мы хотим найти его (ее) адрес. Мы задаем условие поиска следующим образом:

Идентификационный номер = 34924
Фамилия =

Поскольку в структуре Записей учета продаж отсутствует поле с наименованием Фамилия, Записи этого типа не будут просматриваться системой.

Если мы хотим найти Записи учета товарно-материальных ценностей, то мы должны организовать поиск по условию:

Идентификационный номер = 34924.
Дата =

Будут найдены Записи именно этого типа, так как только они содержат поле "Дата". Все это довольно просто объяснить и не так уж сложно программировать. При этом надо иметь в виду, что конечный пользователь не желает запоминать ярлыки "Идентификационный номер", "Дата" и др. Он (или она) хотел бы видеть интересующую его информацию в виде формы. Но для этого необходимо, чтобы ярлыки и поля были связаны между собой.

4.1. РЕЛЯЦИОННАЯ АЛГЕБРА

База данных, в которой Записи связаны друг с другом посредством общего поля, на профессиональном жаргоне называется "реляционной"*. Происхождение связано с разделом математики, называемым "реляционная алгебра".

Ниже приводится краткое описание реляционной модели данных с тем, чтобы читатель понимал, что имеется в виду, когда специалисты говоряд о реляционной базе данных.

Записи о покупателях

Имя	Фамилия	Адрес 1	Адрес 2	Адрес 3	Идентификационный номер
Фред	Джонс	35	Хайстрит	Оулдхэм	24251
Мери	Смит	Ларчис	Магнолия стрит	Кингстон	34294

* Некоторые люди полагают, что термин "реляционный" означает, что все элементы информации в Записи связаны между собой, и что если Вы выбираете один элемент, то одновременно получаете доступ ко всем остальным элементам. Внешне это выглядит таким образом, но подобное объяснение мало что дает, так как невозможно представить, как еще может быть осуществлена Запись. Поэтому употребление слова "реляционный" в таком смысле может вызвать непонимание. Ясное и четкое изложение теории баз данных можно найти в [1].

Записи учета продаж

Идентификационный номер	Дата	Буквенный код	Инвентарный номер
34924	24 фев. 1984	aac	66831
22726	16 май 1984	aab	55673
34924	23 июнь 1984	ada	29871

Математик скажет, что здесь представлены два отношения, имеющие общий элемент – значения в столбце с именем "Идентификационный номер". Операции реляционной алгебры могут объединить два типа Записей по общему элементу "Идентификационный номер".Тогда Запись, скажем о Мери Смит, запишется так:

Мери Смит Ларчис Магнолия стрит Кингстон 34924 24 фев. 1984 aaa 66831

Другими словами, к сведениям о личности покупательницы и ее адресе будут добавлены сведения об одной из сделанных ею покупок.

Если объединить Записи, хранящиеся в базе данных, по буквенному коду и инвентарному номеру, то мы узнаем, что было куплено и у кого.

Элементы данных, связывающие вместе две Записи, могут быть уникальными для данной пары, но могут повторяться и во многих Записях, как, например, повторяется идентификационный номер Мери Смит. Они могут повторяться неоднократно и связывать Записи различных клиентов или Записи одного и того же типа. В гл. 3 была описана база данных политических новостей. В этой базе каждое сообщение связано с предыдущим событием и в дальнейшем будет связано с новым событием.

В основе хорошего пакета программ управления реляционной базой данных должна быть программа, которая просчитывает число связей и следит за тем, чтобы ни одна Запись не осталась "сиротой". Допустим, в базе данных хранятся Записи двух типов: о покупателях и хозяйственных операциях. Записи о хозяйственных операциях имеют смысл только тогда, когда они связаны с Записями о покупателях. Если последние исчезнут или в них изменятся идентификационные номера, то Записи о хозяйственных операциях станут "бесхозными". Чтобы избежать подобной ситуации, СУБД должна постоянно пересчитывать число связей в каждом направлении: одна Запись о покупателе может указывать на 15 различных хозяйственных операций (сделок), в то время как каждая из 15 Записей сделок должна указывать на одного покупателя. Чтобы это проверить, просчитывают число связей от одной Записи к другой, а затем ведут счет в обратном напралении, полученные суммы должны совпадать.

Еще одна операция реляционной алгебры, которую пользователи стремятся выполнить над одним или более типами Записей в реляционной базе данных, это "Проекция". Представим себе, что мы продаем пишущие машинки по заказам, поступающим по почте. В базе данных хранятся Записи о клиентах, хозяйственных операциях и товарно-материальных ценностях. Мы хотим знать, пишущие машинки каких марок покупают клиенты в Уоллсолле. Если мы отберем Записи клиентов, проживающих в Уоллсолле, и объединим их с Записями о хозяйственных операциях, а затем снова

объединим с Записями учета товарно-материальных ценностей и выберем данные о поставщиках, мы получим ответ на свой вопрос. Допустим, выяснилось, что мы продали пишущие машинки только двух марок: не теряющую популярность "Гипертронику" и ИБМ . Полученные с ЭВМ данные выглядят таким образом:

Покупатель	Марка пишущей машинки
Смит	Гипертроник
Джонс	Гипертроник
Блэк	ИБМ
Уайт	Гипертроник
Аллен	ИБМ
Акеншоу	ИБМ

Нам нужно получить из ЭВМ всего две строчки. Мы же получили несколько записей, по одной на каждого клиента, которые надо рассортировать по маркам пишущих машинок, т. е. собрать все "Гипертроники" вместе, а затем то же самое проделать с ИБМ .

Покупатель	Марка пишущей машинки
Смит	Гипертроник
Джонс	Гипертроник
Уайт	Гипертроник
Блэк	ИБМ
Аллен	ИБМ
Акеншоу	ИБМ

После чего копировать столбец с именем "марка пишущей машинки", сравнивая каждую запись с предыдущей, и если записи совпадают*, переходить к сравнению следующей записи с предыдущей. В конечном итоге мы получим: "Гипертроник", ИБМ , т. е. интересующий нас список.

При поверхностном рассмотрении может показаться, что применение к Записи базы данных операций реляционной алгебры в значительной степени упрощает задачу программиста, так как этот раздел математики хорошо разработан и понятен, по крайней мере, математикам.

К сожалению, реляционная алгебра – еще один пример (из числа примеров, столь распространенных в научно-техническом мире), когда под существующий метод подбирается задача. Реальные данные не укладываются точно в абстрактные множества. Основная проблема заключается в том, что "отношения" могут содержать повторяющиеся элементы.

* Еще одна причина, по которой хороший пакет программ управления базой данных должен выполнять операции проверки записей. Записи, если они предполагаются одинаковыми, действительно должны быть таковыми.

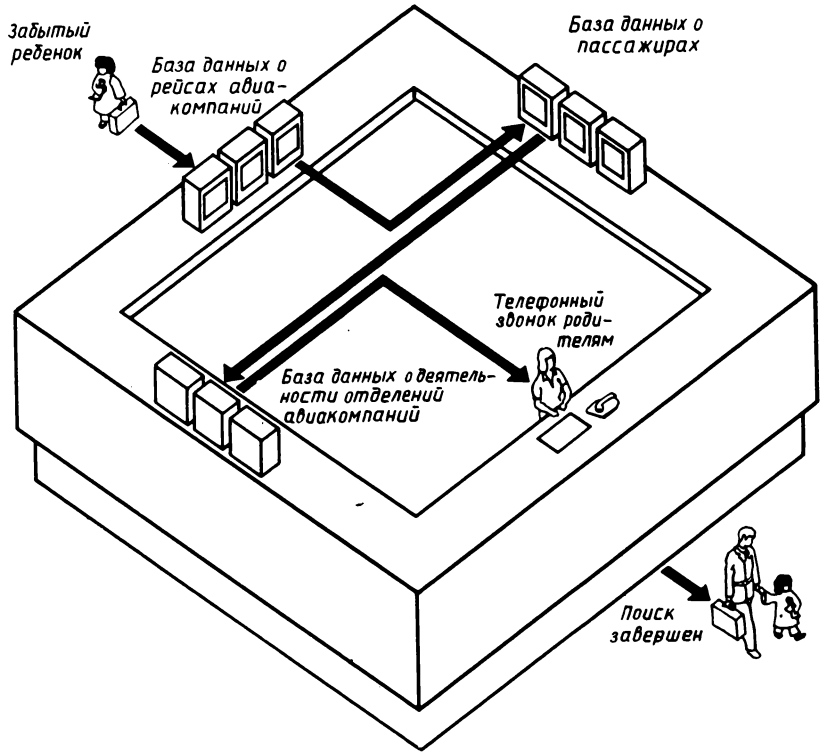


Рис. 34. В аэропорту обратили внимание на маленькую девочку-китаянку, которую никто не встретил. Поиск в базе данных о рейсах авиакомпании показал, что она могла прибыть рейсом из Сингапура или Гонконга, наиболее вероятных мест вылета. Просмотр списков пассажиров позволил выявить, что одним из этих рейсов летела девочка примерно того же возраста, без сопровождения взрослого. Телефонный звонок в отделение авиакомпании, где продали билет, позволил установить фамилию и имя родителей девочки. После телефонного разговора с родителями стала известна фамилия человека, который должен был ее встретить

Однако, как мы видим, идея использования повторяющихся элементов в записях очень заманчива и перспективна. Она порождена здравым смыслом и практическим опытом людей. Более того, обычные пользователи ЭВМ не знают реляционной алгебры. Операции объединения и проекции записей ясны и понятны математикам, но они не интересуют пользователей.

Любой пользователь ЭВМ может понять из практики, как общие элементы данных могут связывать две или более записи. Но зачем усложнять решение практических задач теоретическими положениями, без которых вполне можно обойтись. То, что задачи реальных баз данных имеют сходство с задачами, для которых применимы методы теории множеств,

сыграло злую шутку: в соответствии с "законом Паркинсона" появилась совершенно абстрактная дисциплина, которая широко распространилась в интеллектуальном мире.

Все, чего с помощью этой дисциплины можно добиться, это запутать реальные проблемы трудными и ненужными теоретическими выкладками. Их знание можно проверить на экзаменах, но в решении практических задач они не нужны.

Возможно, мои высказывания звучат слишком категорично, но у меня есть к тому основания. Мы начали разрабатывать систему *Superfile*, блуждая в потемках невежества. Не зная, как работают базы данных, мы создали систему, основываясь на возможностях микроЭВМ выпуска 1981 г. Система выполняла полезные, нужные пользователю операции. По окончании разработок потенциальные покупатели попросили сравнить нашу систему с существовавшими в то время пакетами программ управления базами данных. Я должен признать, что мы оказались в затруднительном положении. Мало что можно было сравнивать (как нам казалось), поскольку традиционные разработки оказались такими сложными, что их трудно было освоить и применять. Мы не могли тогда понять, откуда возникли проблемы, которые пытались решить их разработчики. Время и опыт ответили на этот вопрос. Корни трудностей уходят в прошлое. Первые коммерческие компьютеры хранили долгосрочную информацию на магнитной ленте, так же как современные бытовые ЭВМ хранят ее на звуковых кассетах. Любопытно, но накопитель на магнитной ленте стал стандартным зрительным символом вычислительной техники в фильмах и на экране ТВ, хотя он уже почти не используется в составе ЭВМ.

Каждый, кто пытался использовать кассету в качестве машинного носителя информации, знает, что у кассеты есть два серьезных недостатка: Вы можете считывать ленту только от одного конца к другому (время выборки данных с ленты различное, в отличие от диска); очень трудно остановить ленту точно в заданном месте, чтобы считать необходимую информацию. Эти недостатки магнитной ленты заставляли первых разработчиков баз данных пускаться на удивительные ухищрения при обработке данных: копировать записи постоянной длины из одного файла в другой, проводить слияние массивов, их сортировку, добавление записей. На мой непросвещенный взгляд, они решали необыкновенно трудные задачи, вроде запутанных головоломок.

И как это часто бывает, временные недостатки техники породили длительные проблемы теории. Разработчики баз данных, преодолев с большим трудом недостатки накопителей на магнитных лентах, перестали замечать, что их решения относятся уже к несуществующим проблемам, что дисковая память свободна от этих ограничений. Мне кажется, многие трудности, которые испытывает потребитель с традиционными базами данных, вызваны архаичными методами, унаследованными от старой технологии.

Напрашивается сравнение с капитаном первого парового судна, который настаивает на том, чтобы его корабль был отбуксирован в открытое

море, где можно "поймать ветер". Всю свою жизнь он обучался искусству управления парусами и не хочет отказаться от своих знаний и навыков из-за того, что паруса устарели. Однако вернемся к теме книги.

Реляционный подход легко объяснить, но довольно трудно осуществить при программировании. Все связующие элементы должны быть индексированы, если мы хотим, чтобы поиск в базе данных успешно осуществлялся от одной Записи к любой другой Записи.

Такая организация поиска в базе данных является самой сложной: она осуществляется по принципу "многие – многие", т. е. многие Записи одного типа могут быть связаны с многими Записями другого типа. Для сравнения отметим, что в иерархической базе данных поиск осуществляется по принципу "один – многие": каждая родительская Запись может иметь многие дочерние Записи. Последние, в свою очередь, могут иметь свои дочерние Записи, но каждая дочерняя Запись может быть связана только с одной родительской Записью.

Связи встроены в систему, и поиск обычно может идти только по одному пути – от родительской Записи вниз. Все это хорошо, если поиск в другом направлении Вам не нужен. Но реализовать в иерархической базе запрос на поиск дефектных пишущих машинок в Кингстоне просто невозможно.

Если Вы возьмете за основу полностью индексированный плоский файл, то Вы сможете довольно быстро создать реляционную базу данных. Но как обеспечить достаточное время обработки запроса системы, если поиск ведется по многим признакам, а связи организованы по принципу "многие – многие"?

В больших базах данных связи часто создаются явно: Запись содержит ссылки на другие Записи, и можно обойтись без механизма индексирования. Заманчивое решение, но оно означает, что:

1) Вы знаете заранее, с какими запросами будут обращаться к базе, например: найти всех покупателей в Кингстоне, которые приобрели дефектные пишущие машинки марки "Гипертроник";

2) у Вас есть время просматривать базу данных вдоль и поперек и заранее вставлять необходимые указатели. Эта операция столь же трудоемкая, как и индексирование, и выполняет подобную функцию;

3) имеется свободная дисковая память, достаточная для размещения указателей;

4) после того как указатели введены в базу данных, информация в базе может измениться, но это не отразится на указателях.

Чтобы избежать подобных ситуаций, в программном обеспечении баз данных предусматривается, что при удалении из базы данных Записи "Б", на которую имеется ссылка в Записи "А", Запись "Б" физически остается на прежнем месте, но в ней появляется указатель на новый вариант Записи "Б". Спустя некоторое время в базе данных образуется сложная сеть указателей, в том числе на удаленные Записи.

Некоторые СУБД, разработанные для больших ЭВМ с учетом этих

принципов, стали настолько сложными, что требуются значительные затраты времени высококвалифицированных системных программистов, чтобы вновь перезагрузить базу данных, другими словами очистить ее от ненужных Записей и связанных с ними указателей на указатели.

4.2. ИЕРАРХИЧЕСКАЯ БАЗА ДАННЫХ

Иерархическая база данных проще по своей организации, но и возможности ее меньше по сравнению с реляционной базой данных.

В иерархической модели организации данных каждая Запись имеет единственное поле индекса – ключевое поле. В зависимых Записях тем или иным способом повторяются ключи родительских Записей. Эти связи различаются системой, но они невидимы для пользователя (напомним, что общие элементы в Записях реляционной базы, устанавливающие между ними связи, видны пользователю). Индексирование намного упрощается, поскольку индексируются только корневые Записи. Для некоторых приложений это идеальная модель. Многие документы имеют иерархическую структуру: при поступлении пациента в больницу делается запись, в которой указываются фамилия больного, домашний адрес, дата госпитализации, фамилия лечащего врача, номер палаты. Больной затем несколько раз посещает рентгеновский кабинет. Ему назначают лекарства. Он ходит на физиотерапию. В лаборатории ему делают анализы. Если ничего не помогает и затем следует смерть человека, то делается Запись о летальном исходе.

Все эти записи хорошо укладываются в иерархическую структуру. Каждый раз создается запись различного типа, но все они связаны с основной записью. Структура данных получается древовидной.

Однако преимущества иерархической базы кончаются, как только Вы захотите выявить пациентов, которым сделали инъекции из негодной партии пенициллина. В реляционной базе данных Вы ведете поиск по номеру партии пенициллина, а затем по идентификационным номерам находите пациентов, которым сделали инъекции этого пенициллина. В иерархической базе этого сделать нельзя. Вы должны последовательно перебрать корневые записи всех пациентов, просмотреть каждое дерево, идущее от корневой Записи и найти Запись, содержащую номер негодной партии. Коли-

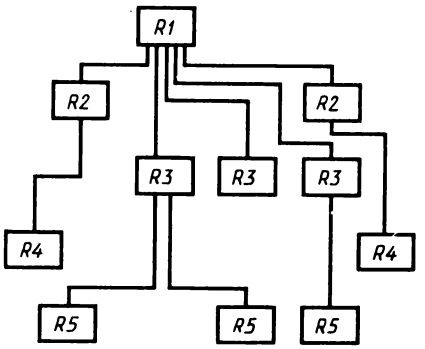


Рис. 35. В иерархической базе данных "зависимые" Записи связаны с корневыми Записями. Такая структура может повторяться. Например: R1 – корневая Запись, R2 и R3 – подчиненные ей Записи. Они же, в свою очередь, являются корневыми Записями для R4 и R5. Безусловно, каждый тип может повторяться несколько раз

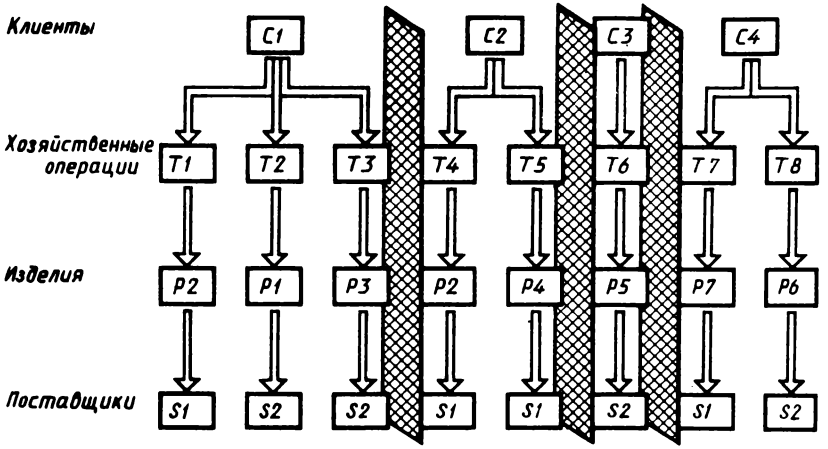


Рис. 36

чество операций, связанных с поиском, гораздо больше, чем в реляционной модели данных.

Но дальше еще хуже. Недостатки иерархической базы данных хорошо видны, если мы перерисуем схему реляционной базы данных и перейдем к иерархической.

В этом случае требование о том, чтобы каждая зависимая Запись имела одну родительскую Запись, легко выполняется на первых двух уровнях. Записи о сделках покупателей, конечно, зависят от Записей о самих покупателях. А Записи об изделиях? Несколько покупателей могут приобрести одни и те же изделия. Это означает, что в иерархической схеме могут неоднократно повторяться одинаковые Записи, например изделие 2 встречается дважды, поскольку оно было продано при совершении сделок 1 и 4. На следующем уровне получается еще хуже. Два наших поставщика, для которых раньше было достаточно двух Записей, теперь требуют восемь Записей. Здесь мы не только растрачиваем дисковую память, сама эксплуатация базы данных становится почти невозможной: если кто-либо из поставщиков сменит адрес, надо перезагрузить всю базу данных, внеся изменение в неизвестное число Записей.

Еще одна трудность, связанная с иерархической базой данных, заключается в том, что в ней не так просто хранить сведения о поставщике, который нам еще ничего не продал. Чтобы это сделать, надо присоединить Запись о таком поставщике к Записи о несуществующем клиенте. Вообще говоря, если Вы можете реализовать реляционную модель данных, незачем заниматься иерархической моделью. Реляционная модель данных всегда внешне может быть представлена как иерархическая, но не наоборот.

Наконец, есть еще сетевая модель организации данных, в которой связи между Записями сами составляют Запись некоторого типа. Один из

разработчиков пакета программ Unix однажды сказал: "Сетевая база – это самый верный способ потерять данные". Этим объясняется то большое внимание, которое разработчики Unix уделили проверяемым древовидным структурам для хранения файлов. Эти файлы во многих отношениях похожи на Записи в базе данных. Однако пакеты программ управления сетевыми базами данных еще не появились для микроЭВМ, поэтому мы не станем рассматривать эту модель базы данных.

4.3. ОТОБРАЖЕНИЕ СВЯЗАННЫХ ЗАПИСЕЙ НА ЭКРАНЕ

Для того чтобы отобразить многофайловую базу данных на экране, мы должны показать конечному пользователю сразу несколько выходных форм. Современные микроЭВМ обладают одной интересной особенностью: Вы можете разбивать экран на несколько отдельных экранов, так называемых окон. Обычно, когда выполняются различные программы (например, редактор текста, программа расчета рабочих таблиц, программа построения графиков), результаты работы каждой отображаются в своем окне. Выглядит это так, словно три отдельных дисплея соединены с тремя самостоятельными микропроцессорами.

Мы же хотим в каждом окне выводить различную форму, но они должны быть связаны между собой невидимым для конечного пользователя способом так, чтобы все поля и их значения автоматически считывались при переходе от одной формы к другой. Формы, соответствующие индивидуальным записям, составляют суперформу точно так же, как отдельные плитки составляют мозаичный пол. В суперформу переносится все, что появляется между элементами – связками одной формы и соответствующими элементами другой формы. Может быть скопировано именованное поле и его значение, а если значение отсутствует, то просто именованное поле с тем, чтобы облегчить поиск. Проще пояснить этот процесс, обратившись к рисунку, чем стараться описать его словами.

а) Мы хотим найти клиента из Кингстона (окно *1*), который купил пишущую машинку "Гипертроник" (окно *3*). Вводим эти два условия.

б) Система управления реляционной базой данных выбирает первого клиента в Кингстоне и печатает ее идентификационный номер. С помощью идентификационного номера система выбирает первую сделку этого клиента (окно *2*), от которой программа переходит к просмотру Записи о пишущей машинке "Гипертроник". К сожалению, ивентарный номер не совпадает с номером, указанным в запросе.

в) Система продолжает поиск и выбирает Запись другого клиента в Кингстоне.

г) На этот раз Запись о хозяйственной операции содержит инвентарный номер, который соответствует номеру в запросе.

Суперформа по своей сути – это единый документ. Пользователь может ее редактировать так же, как он редактирует текст с помощью программы-редактора. Разница заключается лишь в том, что курсор можно перемещать только в пределах полей формы, а не в любое место экрана.

Окна, содержащие отдельные формы, не обязательно имеют обозначенные на экране границы. Их можно объединять. Например, при составлении транспортных документов используется информация из Записей о покупателе, хозяйственной операции и учете товарно-материальных ценностей. Мы знаем, что эти данные хранятся в различных местах базы данных, но наши пользователи не имеют и не хотят иметь об этом ни малейшего представления. Все, что они хотят, это аккуратная привычная для глаза накладная, при виде которой они достают свои авторучки и выписывают денежный чек — ради чего все эти операции и выполняются.

```
Форма "Покупатель"
            Имя: [ Мэри]  Фамилия: [ Смит]
            Адрес: [ Ларчис]
                   [ Магнолия  стрит]
                   [ Кингстон]
--------    Идентификационный номер: [ 34294] ----------
Форма "Сделка"
            Дата: [ 24 февраля 1984]
--------    Изделие: Буквенный код: [ aac] Инвентарный номер
                66831] ------------------------
Форма "Изделие"
            Описание: [ Электрическая пишущая машинка марки
                  "Гипертроник"]
            Поставщик: [ Гипоид Тайпрайтер]
            Тип изделия: [ ПЛ34251]
            Серийный номер: [ 229221]
            Цена: [ 221.67]
```

Рис. 37. Связанные Записи представлены в различных окнах экрана. Элемент Идентификационный номер объединяет первые два типа Записей. Буквенный код и Инвентарный номер объединяют Записи второго и третьего типов. Хотя элемент-связка присутствует в каждой из объединяемых Записей, он показывается на экране один раз и принадлежит сразу обоим окнам

```
Форма "Покупатель"
            Имя: [      ] Фамилия: [          ]
            Адрес: [          ]
                   [          ]
                   [ Кингстон ]
---------   Идентификационный номер: [          ] ----------
Форма "Сделка"
            Дата: [          ]
---------   Изделие: Буквенный код: [     ] Инвентарный номер: [     ]
Форма "Изделие"
            Описание: [ Гипертроник                          ]
            Поставщик: [                          ]
            Тип изделия: [                        ]
            Серийный номер: [                     ]
            Цена: [          ]
```

а)

Форма "Покупатель"
Имя: [Мэри] Фамилия: [Смит]
Адрес: [Ларчис]
[Магнолия стрит]
[Кингстон]
----------Идентификационный номер: [34294]----------
Форма "Сделка"
Дата: [ ]
----------Изделие: Буквенный код: [ ] Инвентарный номер: [ ]
Форма "Изделие"
Описание: [Электрическая пишущая машинка марки
"Гипертроник"]
Поставщик: [Гипоид Тайпрайтер]
Тип изделия: [ПЛ34251]
Серийный номер: [229221]
Цена: [221.67]

б)

Форма "Покупатель"
Имя: [Дж] Фамилия: [Джонс]
Адрес: [2]
[Хай стрит]
[Кингстон]
----------Идентификационный номер: [29368] ----------
Форма "Сделка"
Дата: [03 марта 1984]
----------Изделие: Буквенный код: [aad] Инвентарный номер
[97642] ----------
Форма "Изделие"
Описание [Электрическая пишущая машинка
марки "Гипертроник"]
Поставщик: [Гипоид Тайпрайтер]
Тип изделия: [ПЛ34251]
Серийный номер: [229221]
Цена: [221.67]

в)

Форма "Покупатель"
Имя: [Мэри] Фамилия: [Смит]
Адрес: [Ларчис]
[Магнолия стрит]
[Кингстон]
----------Идентификационный номер: [34294] ----------
Форма "Сделка"
Дата: [24 февраля 1984]
----------Изделие: Буквенный код: [aac] Инвентарный номер
[66831] ----------
Форма "Изделие"
Описание: [Электрическая пишущая машинка марки
"Гипертроник"]
Поставщик: [Гипоид Тайпрайтер]
Тип изделия: [ПЛ34251]
Серийный номер: [229221]
Цена: [221.67]

г)

Рис. 38.

4.4. КОРРЕКТИРОВКА ДАННЫХ

Как это часто бывает в области вычислительной техники, серьезные проблемы скрываются за видимой частью работы системы. Одна из них – корректировка Записей в базе данных. Рассмотренная нами реляционная база данных хранит определенное число Записей, которые могут быть связаны друг с другом с помощью одинаковых элементов в Записях. Мы можем различить тип Записи по уникальным именованным полям, которые не имеют ничего общего с другими полями. Пользователь может изменить любой элемент данных в любой Записи и в любой момент времени. В этом нет ничего сложного, если изменения вносятся в поле адреса или итоговое поле инвентарного учета. Совсем другое дело, когда меняется значение элемента – связки. Если мы меняем значение элемента "Идентификационный номер" в Записи о покупателе, и не меняем в Записи о хозяйственных операциях, база данных будет разрушена, в ней будет храниться информация о клиентах, которые не совершают никаких покупок, и информация о хозяйственных операциях, неизвестно кем совершенных.

Не надо надеяться на то, что никому не потребуется менять идентификационный номер. "Закон Мерфи" проявляется в электронной обработке данных с особой силой: можно со 100 %-ной уверенностью утверждать, что такие ошибки будут сделаны.

Вот почему СУБД должна включать программы проверки связей, установленных в базе данных. Существующие методы контроля различаются по сложности. Один из самых простых методов заключается в том, чтобы постоянно просчитывать число связей для каждого типа Записей. Например, если покупатель сделал 10 покупок, Запись по каждой из них содержит его идентификационный номер. Система записывает рядом со значением идентификационного номера число 10 в Записи о покупателе и число 1 в записях о хозяйственной операции. Система должна подсчитать, что таких записей о хозяйственных операциях в базе данных 10. Функцию проверки связей выполняет специальная программа, которая следит, чтобы в системе не оказалось не подтвержденных связей. Если такие односторонние связи выявлены, пользователь должен принять меры к исправлению данных в базе.

Внести исправление не так уж сложно, раз мы признаем, что это необходимо. Однако существует источник более скрытых ошибок.

Предположим, что у нас есть база данных о покупателях, продажах и складских запасах (довольно типичный случай). Мы вводим информацию о том, что клиент (сведения о нем хранятся в Записи о покупателе) приобрел в нашем магазине разводной гаечный ключ. Описание этого изделия уже хранится в базе данных вместе с наименованием фирмы-изготовителя. Необходимо внести изменения в базу данных: добавить Запись о продаже, изменить дебет у покупателя и вычесть единицу из общего числа разводных гаечных ключей, хранимых на складе.

Вот здесь и возникает проблема. Хозяйственная операция ведет к изменению Записи о складских запасах. Любому из нас понятно, что необ-

ходимо уменьшить общее число гаечных ключей на единицу. Но для СУБД это лишь замена символа в Записи. Допустим, что ошибочно оператор изменил и наименование фирмы-поставщика. Теперь гаечные ключи изготовляет уже не фирма "Снукс энд Компани", г. Хаддерсфилд, а фирма "Кристиан Диор", г. Париж. Для СУБД это опять лишь изменение нескольких символов, но зато какая большая разница для отдела материально-технического снабжения, поскольку "Кристиан Диор" не продает гаечных ключей.

Как можно программным путем выявить такие случайные ошибки и не допустить, чтобы они разрушили всю систему? Ответ непростой. Чтобы обеспечить подобный контроль, требуется сложный механизм ключевых полей, схем и подсхем, словарей и т. д. (см. гл. 5). Все это необходимо для больших ЭВМ и больших баз данных, в которых доступ к данным и внесение изменений осуществляют работники различных отделов. Они не знают и не хотят знать, как работает система в целом. Возможно, вне вычислительного центра нет ни одного человека, который знает, как работает база данных — и даже у специалистов центра нет уверенности, что они это знают. К счастью, СУБД для микроЭВМ находятся в более заботливых руках, чем системы для больших ЭВМ. Для пользователя микроЭВМ очевидно (или должно быть очевидно), что идея замены поставщика является далеко не самой удачной. Другое дело, если поставщик действительно сменил адрес. Соответствующую Запись в базе данных необходимо изменить. Если принять простые меры предосторожности против опечаток и использовать базу данных понятной структуры, то можно надеяться, что обычный здравый смысл поможет пользователю решить эти проблемы.

С философской точки зрения подобные проблемы вызваны попыткой моделировать на ЭВМ объекты реального мира. Для настоящей фирмы "Снукс энд компани" из Хаддерсфилда, с ее печами и персоналом молчаливых йоркширцев, превращение в фирму "Диор" из Парижа было бы сложной задачей. Такое превращение не могло быть вызвано неосторожным нажатием клавиши. Даже для того чтобы изменить запас гаечных ключей на складе, в реальных условиях кто-то должен подойти к полкам на складе, взять ключ и потерять его. Разводные гаечные ключи, как известно, это тяжелые металлические предметы. Не так-то просто их потерять. А вот несколько намагниченных участков на поверхности диска, которые описывают гаечные ключи, потерять очень легко.

ГЛАВА 5
ЯЗЫКИ ЗАПРОСОВ

Пока мы вели разговор о довольно простой обработке информации в базе данных. Мы исходим из того, что после передачи данных программе через формы, отображенные на экране, со значениями полей Записи можно выполнять расчеты. При генерации отчета также можно проводить вычисления внутри Записи, находить промежуточные и общие итоги.

Хотя решение таких задач на основе базы данных очень полезно, этим не исчерпываются все приложения базы данных. Часто надо решать более сложные задачи, и во многих пакетах программ предусматриваются специальные языки, чтобы программировать такие задачи. Допустим у нас есть автоматизированная система учета кадров, в которой применяются следующие правила оплаты листков по временной нетрудоспособности:

1) если работник болен менее 3 дней, то за каждый день пособие выплачивается в размере его обычной дневной ставки при условии, что общее число дней, потерянных по болезни в этом году не превышает 20;

2) если работник болен от 3 до 40 дней (без учета общевыходных и праздничных дней), фирма прибегает к услугам государственной системы социального страхования;

3) если работник болен свыше 40 дней, то вступает в действие система коммерческого страхования.

Еще пример. Мы разрабатываем систему учета товарно-материальных ценностей для магазина одежды. Детскую одежду размером 10 и ниже в основном покупают небольшого роста, изящные и в то же время экономные женщины. Эта одежда не облагается 15 %-ным налогом. Программы управления базой данных должны различать следующее:

```
IF РАЗМЕР < 12 THEN НАЛОГ = 0
ELSE НАЛОГ = 0,15
```

Или, скажем, разрабатывается система для международной страховой компании. Если работник занимает должность "директора регионального отделения", он имеет право на получение 2,36 % суммы комиссионных, выплаченных его подчиненным. Руководитель, отвечающий за деятельность компании в нескольких районах, получает 3,96 %. Расчеты становятся более интересными, если статус руководителя постоянно меняется. Один и тот же человек может занимать скромную должность руководителя отдела сбыта в США и в то же время быть управляющим фирмы "Пан Галактик Гипэр" за границей и получать надбавку 6,1 %. Догадливый программист обратится к этой фирме с просьбой развернуть свою деятельность повсюду, но не все трудности программирования разрешаются таким прямолинейным путем. Итак, нужны программы, позволяющие решать такие задачи. Это означает, что нужны некие языки программирования. Термин "язык запросов" является общим для таких языков. Язык запросов должен обладать рядом возможностей, характерных для всех языков программирования, а именно:

1) присваивать переменным их значения – числа, текст и т. д. В алгебре x и y заменяют числа, которые в данный момент неизвестны. Тот же принцип применяется в языках программирования, только имена переменных более информативны, например "Дневная ставка" или "Размер",

2) проводить расчеты и манипуляции с переменными:

```
СКОРОСТЬ = РАССТОЯНИЕ/ВРЕМЯ;
```

3) проверять условия и передавать управление –
IF СКОРОСТЬ < 30 THEN PRINT "Движется слишком медленно"
ELSE PRINT "Доставит Вас своевременно".

Это одна из важнейших характеристик языка программирования. Она позволяет программе менять направление действий в зависимости от результатов расчетов. Достигается это проверкой переменной – в данном случае с именем "СКОРОСТЬ": имеет ли она значение меньше 30. Если да, то выполняется оператор PRINT "Движется слишком медленно". Если скорость не менее 30, то программа просматривает следующий оператор ELSE, который говорит ЭВМ, что именно делать в этом случае: выполнять оператор PRINT "ДОСТАВИТ ВАС СВОЕВРЕМЕННО".

4) организовать циклы – другими словами, повторять одну и ту же операцию несколько раз. Например, мы хотим увеличить цены всех товаров, хранящихся на складе, на 10 %. В программе организуется цикл, в течение которого извлекается Запись по каждому товару и в нее переписывается новая цена. Фактически цикл включает проверку условия и передачу управления назад в одно и то же место программы,

5) эти четыре возможности должны быть реализованы в любом языке программирования.

Для работы с базой данных язык должен обладать еще некоторыми средствами манипуляции с элементами данных в Записях, а также включать операции с массивами, вроде объединения массивов в реляционной базе данных.

5.1. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

В состав большинства пакетов программ управления базами данных входят собственные языки запросов. Как правило, они основываются на действующих языках программирования, к которым добавляются специальные команды для выполнения определенных операций в базах данных.

Стоит ли добавлять специальные команды к действующему языку программирования? Ведь это требует изучения нового языка, ведет к появлению большого числа ошибок в программах. Кроме того, где гарантия, что разработчики языка запросов охватят все необходимые Вам функции? Мнения на этот счет расходятся.

В руководстве по *dBaseII*, наиболее широко распространенному пакету программ управления базой данных для микроЭВМ, приводится такой формат команды:

```
          [ALL]
DISPLAY [RECORD n][OFF <expression>]
          [NEXT n]
```

Команда содержит указание машине показать Вам Записи. В соответствии с этим форматом Вы должны набрать на клавиатуре:

```
DISPLAY ALL ITEM,PART:NO,COST*ON:HAND,$(PART:NO1,2)
FOR;COST>100.AND.ON:HAND>2 OFF
```

В результате на экране или распечатке появится такая таблица

ТАНКИ "ШЕРМАН"	89793	404 997,00	89
ТРОМБОНЫ	76767	15076,12	76
КОЛЬЦА ЗОЛОТЫЕ	70296	1000,00	70

В руководстве не объясняется, что происходит по этой команде, но можно предположить, что на экране отображаются названия всех изделий стоимостью более 100 фунтов стерлингов, запас которых превосходит 2 шт., а также номера, цена и фактическое количество на складе.

В *dBaseII* имеется программа, генерирующая на экране форму, с использованием которой эту задачу можно решить более удобным путем. Однако взаимодействовать с ней не так просто и многие пользователи не используют эту возможность.

Ниже приводится фрагмент программы, написанной на специально созданном для *dBaseII* языке программирования "Д":

```
...
DO WHILE.NOT.EOF.AND.£<=RecoCount
    & command
    IF ! (Conditions)>CHR(0)
        IF & Expression
            STORE(Count+ 1)TO Count
        ENDIF
    ELSE
        STORE [Count+ 1)TO Count
    ENDIF
    SKIP ...
```

Опыт работы с ЭВМ показывает, что надежность работы программы, содержащей большое количество пробелов, точек и восклицательных знаков, как это видно даже из небольшого фрагмента программы, очень невелика. Опыт также подсказывает, что руководства по программированию не объясняют всего, что нужно пользователю, и если Вы все же напишете программу, Вас ждут неприятные сюрпризы.

Мне бы не хотелось создавать впечатление, что я специально придираюсь к *dBaseII*. Поэтому приведу пример из другого руководства – *FMS*-80.

```
...
(* поиск текущего инвентарного номера в новом массиве данных
```

```
о поставщиках и включение нового наименования поставщика, если
номер найден *)
        2,2 =1,1                        (* инвентарный номер *)
        Kread 2;
        if error 2                      (* не найдено *)
            3,9 =";                     (* пустая строка*)
        else
            3,9 =2,2
        endif
        write 3
end
```

. . .

Потребителей микроЭВМ пытаются убедить, что это некоторым волшебным образом не языки программирования, и поэтому пакеты программ управления базами данных рассчитаны на наивного и неопытного "неподготовленного" пользователя. На самом деле это такие же языки программирования, как и любые другие, и Вы не решите с их помощью свои задачи, если не обладаете опытом и настойчивостью программиста.

5.2. ПРОГРАММИРОВАНИЕ В СИСТЕМЕ SUPERFILE

Обсуждая вопрос, создавать или не создавать в системе *Superfile* собственный язык запросов, мы пришли к выводу, что язык программирования остается языком программирования, как бы его не называли, и что непрограммисту решить с его помощью сложную задачу не удается. Кроме этого, мы учитывали, что любой программист знает, по крайней мере, один из языков программирования: Бейсик, Паскаль, Кобол, Фортран, Си и др.

Мы посчитали нерациональным создавать новый язык со всеми его неизбежными недостатками, и ограничились использованием в системе *Superfile* действующих, хорошо отработанных языков программирования и даже машинного кода.

С точки зрения завоевания рынка сбыта возможно это было ошибкой, но зато мы избавили себя и своих клиентов от лишних забот.

В обычном языке программирования отсутствуют специальные команды для работы с базой данных, например команда объединения массивов. Но при этом нетрудно написать подпрограмму, которая выполнит эту функцию.

Если мы хотим обращаться непосредственно к базе данных из прикладной программы, написанной на диалоговом языке Бейсик или языке программирования, использующем компилятор, то она должна быть загружена в оперативную память ЭВМ вместе с системой управления базой данных. Операционные системы для шестнадцатиразрядных микроЭВМ имеют команды "Загрузить и оставить в памяти", которые загружают программу, запускают ее и затем переходят к загрузке другой программы. Поэтому при работе с базой данных сначала в память ЭВМ должен быть

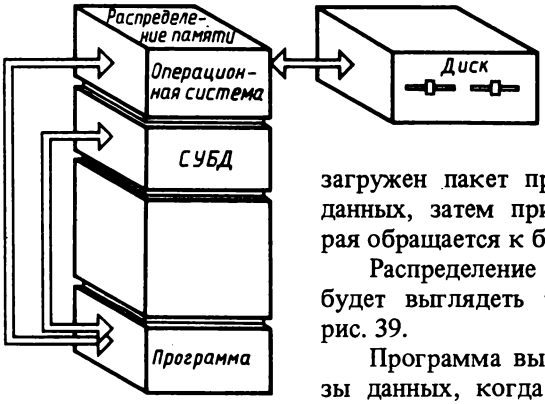


Рис. 39. Распределение памяти микроЭВМ при работе системы

загружен пакет программ управления базой данных, затем прикладная программа, которая обращается к базе данных.

Распределение памяти ЭВМ в этом случае будет выглядеть так, как это показано на рис. 39.

Программа вызывает пакет программ базы данных, когда ей нужна информация из базы, так же как она вызывает подпрограммы операционной системы, когда обращается к файлам. Были расширены возможности операционной системы – она возвращает информацию в виде нужных программе блоков, а не просто файлов. Операционная система находит файл по имени, а запись – по элементам внутри ее. Запись можно рассматривать как короткий последовательный файл, имя которого содержится внутри самого файла.

Такого рода прямой доступ к данным встречается в СУБД для микроЭВМ крайне редко. Поэтому я хотел бы опять вернуться к системе *Superfile*, чтобы объяснить, как нам удалось это сделать. Программа, написанная на Бейсике, использует команду CALL. Строка, содержащая эту команду, выглядит следующим образом:

```
CALL DBM(DF M$(0),R$(0),DR)
```

Команда CALL передает управление по адресу в оперативной памяти, значение которого содержится в переменной DBM. Это точка входа в систему *Superfile*. Команда содержит четыре параметра: DF – число, которое говорит системе, что делать (11 – добавить запись, 9 – найти запись, 12 – удалить запись ...).

Второй параметр M$ (0) – символьный массив, в котором программист размещает текст, засылаемый в базу данных. Если мы хотим найти Запись о клиенте по имени Том Тамбс, то можем написать:

```
M$(0) = "Имя =Том"
M$(1) = "Кредит < 50"
M$(2) = " "
```

Другими словами, мы хотим найти Запись о клиенте, чье имя Том и сумма кредита которого меньше 50 фунтов стерлингов. Если мы задали DF = 9,

то по команде CALL система управления базой осуществляет поиск и вернет интересующую нас Запись в массив R$ ( ):

R$(0) = "Имя =Том"
R$(1) = "Имя Чолмондлей"
R$(2) = "Фамилия =Тамбс"
... .
R$(*n*) = "Кредит =45"
R$(*n* + 1)= "Идентификационный номер =1543"

Что нас в первую очередь интересует, так это сумма кредита. Нетрудно написать короткую программу на Бейсике, которая просмотрит массив R$( ), найдет слово "кредит" в левой части соответствующей Записи и выделит ее правую часть "45".

Объединить эту Запись с Записью о хозяйственных операциях несложно. Программа находит общее поле "Идентификационный номер", извлекает всю строку, переписывает ее в массив R$( ) и ищет в базе данных хозяйственные операции клиента по имени Том. Если мы хотим получить дополнительную информацию, например найти покупки, на которые он тратил более 100 фунтов, мы можем составить запрос таким образом:

M$(0) = "Идентификационный номер =1543"
M$(1) = "Цена > 100"
M$(2) =" "

Необходимо написать подпрограмму, выполняющую функцию объединения двух Записей, которой в качестве аргумента передается имя общего для них поля (в данном случае Идентификационный номер).

Программирование на других языках ведется аналогичным образом.

Обращение из прикладной программы к базе данных – это основа работы с базой данных. Если ядро пакета программ управления базой данных выполняет функции хранения, поиска информации, а другая его часть осуществляет ввод и вывод реальной информации обычно с помощью генерируемых на экране дисплея форм и отчетов, то прикладные программы связывают все эти возможности воедино.

При решении многих простых задач все, что пользователю нужно – это хранить и извлекать данные.

Несложная обработка данных и простые расчеты выполняются программными модулями самой базы данных через интерфейс конечного пользователя. В других случаях выход формируется прикладной программой в виде таблиц, а ввод данных осуществляется с помощью формы на экране дисплея.

5.3. НАГЛЯДНОСТЬ ПРОГРАММ

Программирование отличается от любой другой трудовой деятельности. Чтобы делать это профессионально, надо учиться. Обучение программированию можно сравнить с обучением вождению автомобиля, управлению парусной лодкой, езде на лошади или полету на аэроплане. Требуется от 3 до 6 мес. напряженного труда, чтобы овладеть начальными навыками, и затем несколько лет практики, чтобы научиться делать все профессионально. Некоторым это удается лучше, чем другим, а многие способные и умные люди вообще никогда не смогут научиться.

Тем не менее автору представляется, что программирование несколько усложнено, особенно при написании программных комплексов для управления базами данных. Распространенные языки программирования часто держат программиста в плену традиций, а традиция в программировании означает более примитивные методы обработки данных. Кроме того, все они основываются на алгебре. Программист должен держать в голове сложные структуры переменных и указателей. Язык очень мало помогает программисту манипулировать со сложными логическими построениями в программах. В результате в программах содержится больше ошибок. Развитие языков программирования тормозилось теми средствами программирования, которые были в распоряжении программиста.

До недавнего времени при решении на ЭВМ большинства практических задач использовались перфораторы и телетайпы. Телетайп – это обычная электрическая пишущая машинка: Вы печатаете сообщение для ЭВМ в черном цвете и получаете ответы (обычно в закодированном виде) в красном цвете. Такого рода общение с ЭВМ слишком односторонне, оно не гармонирует с человеческим мышлением и особенно затрудняет понимание того, как в действительности работает база данных. База данных имеет, по меньшей мере, два измерения, а если принять во внимание повторяющиеся поля, то три.

Хотя автор резко высказался в гл. 3 по поводу машинных графиков, современные тенденции в развитии микроЭВМ отчетливо показывают, что люди гораздо полнее усваивают графическую информацию, чем строки текста, заполненные символами. В этом нет ничего удивительного, если вдуматься: картинка несет больше информации, чем эквивалентный ей по площади текст. Хорошо оплачиваемый программист, если у него есть свободное время, может работать с базой данных "вслепую", используя язык программирования. Однако массовое применение вычислительной техники означает, что таких специалистов слишком мало, чтобы мож-

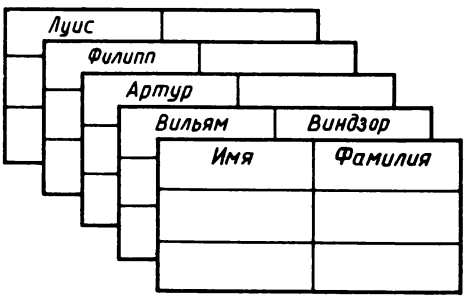


Рис. 40. Повторяющиеся поля делают структуру данных трехмерной

но было надеяться на их помощь. Конечные пользователи должны делать максимум своими руками. Возможно, им легче будет понять работу базы данных, если они будут наблюдать ее визуально.

Самый простой способ наглядно представить себе Запись – это увидеть ее в рамках той же формы, которую мы использовали при вводе информации. Любые необходимые пояснения о том, что происходит, могут быть встроены в форму. В качестве переменных в программе используются поля Записи, а их значения равны содержимому полей. Допустим Вы используете базу данных с информацией о видах транспорта: воздушном, автомобильном, гужевом и т. д. – и в каждой Записи поле 1 содержит наименование вида транспорта, поле 5 – его скорость. Тогда фрагмент программы можно переписать следующим образом:

```
IF F5 <30 THEN PRINT F1 "Движется слишком медленно"
ELSE PRINT F1 "Доставит Вас своевременно"
```

При выполнении программы Вы увидите такие сообщения:
"Велосипед движется слишком медленно".
"Автомобиль доставит Вас своевременно"
"Ослик движется слишком медленно"
"Авиалайнер доставит Вас своевременно"
...

Специфические для базы данных операции, например по объединению массивов, также могут быть отражены визуально. В гл. 4 говорилось о том, как программа – генератор форм, в составе которой есть программа-редактор, позволяет отобразить несколько различных форм – и соответствующие Записи – в различных частях экрана, а затем автоматически связать Записи, переписав содержимое общего для них поля из одной Записи в другую. Это позволяет наглядно представить, как выполняются две обязательные для языков запросов функции: использование переменных и проведение расчетов с их значениями. Для третьей функции – проверка условий и передача управления – визуальный аналог отсутствует. Мы вынуждены эту часть программы писать как обычно. Но, по крайней мере, операторы IF и ELSE могут относиться к видимым полям. Можно отобразить на экране формы и, последовательно показывая все поля, запросить алгоритм, по которому должно рассчитываться значение поля.

Например, мы решаем задачу о больничных листах. В базе данных содержится два типа простых Записей: фамилия служащего, его адрес и данные о заболевании. Они связаны общим полем, содержащим табельный номер служащего, однозначно его определяющий.

Программа – генератор форм автоматически связывает каждую пару Записей в единую виртуальную Запись, и при необходимости мы можем включить величины из любого поля любой формы в нашу расчетную формулу.

Каждое поле обозначается парой букв, начиная с "*aa*" и кончая "*ad*" (поскольку в нашем случае имеется четыре поля). Система помещает

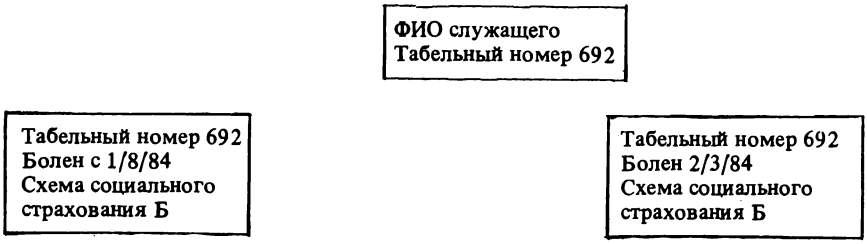


Рис. 41. Запись о служащем и связанные с ней записи о заболеваниях

звездочку в поле "*ad*". Это означает, что пользователь должен ввести алгоритм, по которому рассчитывается значение для этого поля. Наша цель – разделить служащих, имеющих больничные листы, на три группы: с продолжительностью заболевания менее 3 дней (класс А), от 3 до 40 дней (класс В) и более 40 дней (класс С). Мы можем написать:

```
IF Сегодня – AC < 3 THEN AD = "A" OR IF Сегодня – AC > 3
AND Сегодня – AC <= 40 THEN AD = "B" ELSE AD = "C"
```

("Сегодня" – это имя переменной, содержащей текущую дату. Ее значение задается часами, встроенными в ЭВМ). Следующая форма и несколько операторов позволяют продолжить обработку данных, полученных при группировке.

Безусловно, пример выглядит упрощенным. В законодательстве речь идет о рабочих днях. Поэтому необходимо исключить субботы и воскресенья. Это можно сделать, если поделить номер дня на 7. Остаток, равный например 6, может означать вторник. Но в реальной жизни расчеты еще сложнее, так как надо принять во внимание общегосударственные праздники, которые по определению не являются "рабочими днями". Возможно, надо ввести третью форму, позволяющую определить, входят ли в период болезни даты праздников, хранящихся в базе данных. (Все это изложено для того, чтобы показать, что программирование – дело нелегкое, что бы там не говорили).

5.4. ЛОГИЧЕСКИЕ СХЕМЫ И СЛОВАРИ ДАННЫХ

Тому, кто работает с микроЭВМ, приходится сталкиваться с терминами, ведущими свое происхождение от больших ЭВМ, например: схема и словари данных. И хотя лишь немногие СУБД для микроЭВМ используют их в явном виде, полезно знать, что они означают.

Словари данных – это данные о данных [1]. Они содержат описание всех основных объектов системы – типы записей, кто ими пользуется, какие программы к ним обращаются. Назначение словарей данных – оказать помощь администратору базы данных в работе с базой. Если администратор хочет изменить конкретную Запись, он узнает из словаря данных, в какие программы надо внести изменения и каких пользователей надо предупредить об этом.

Понятие "логическая схема" связано с идеальным представлением хранящихся в базе данных, не зависящим от конкретной машины. Она описывает формат Записей, типы данных, коды доступа и т. п. Логическая схема является частью словаря данных. "Внутренняя схема" полностью зависит от конкретной ЭВМ. Она описывает, как данная ЭВМ реализует логическую схему, и содержит размеры полей, систему индексирования, хеширования, указатели, т. е. многие элементы базы, реализуемые системой автоматически и часто невидимые каждому пользователю.

5.5. ЯЗЫК ПРОЛОГ

Неисчерпаемым источником развлечения для специалистов в области вычислительной техники являются споры о языках программирования. Что лучше – Бейсик или Паскаль? Что предпочтительнее? Кобол или смерть? Такие споры – пустая трата времени. Несколько большее основание для спора дает сравнение преимуществ двух языков: языка фон Неймана и языка обработки списков. Первый построен на тех же принципах, на которых основываются языки Бейсик, Паскаль, Фортран, Си. Имеется перечень зарезервированных слов, которые воспринимаются ЭВМ как команды. Наиболее развитые языки позволяют программисту писать "процедуры", т. е. подпрограммы, которые могут быть использованы другими процедурами, а самые изощренные языки допускают "рекурсию", или обращение процедуры к самой себе. Подход фон Неймана к программированию основывается на убеждении, что программист знает в деталях, что он хочет сделать. Он доводит программу до необходимой степени детализации и на этом останавливается.

Язык обработки списков основывается на совершенно другом подходе. Вы начинаете программу с некоторого самого общего предложения, вроде:

"Смысл Вселенной (Боги, Люди)".

Затем Вы пытаетесь продолжить такое удачное начало, добавив несколько дополнительных предложений:

"Природа (Люди, смертные)".
"Природа (Боги, бессмертные)".

Таким путем Вы продолжаете до тех пор, пока ЭВМ не сможет реализовать простейшие выражения вроде 2 + 2 = 4 самостоятельно. На этом составление программы заканчивается. Если все связи в этой цепи указаны правильно, то программу можно выполнить на ЭВМ и, возможно, Вы узнаете смысл Вселенной.

Цель этого небольшого отступления – познакомить читателя с Прологом, языком обработки списков, пользующимся большой популярностью среди специалистов по искусственному интеллекту. Этому языку отдают предпочтение японские специалисты. Говорят, это объясняется тем,

что кто-то сказал, будто Пролог – это формализованный английский язык, и если Вы понимаете Пролог, то можете понимать и по-английски.

В программе на языке Пролог Вы записываете отношения, например:

"Нравится (Джо, рыба)"

"Нравится (Джо, Мери)"

"Нравится (Мери, книга)"

"Нравится (Джо, книга)"

После чего Вы можете задавать ЭВМ вопросы (ответы выделены жирным шрифтом).

"? – нравится (Джо, деньги)"

"Нет"

"? нравится (Мери, книга)"

"Да"

и так далее.

Конечно, пример очень упрощенный. Язык становится полезным и интересным, когда формируются сложные запросы, например: найти того кому нравится тот, кто любит книги и рыбную ловлю и не нравится тот, кому не нравится рыбная ловля, но кто обожает футбол.

Хотелось бы подчеркнуть, что Пролог является очень мощным языком запросов для реляционной базы данных, но в большинстве приложений он используется без пакета программ управления базой данных. Как правило, вся информация, с которой работает программа, написанная на языке Пролог, размещается в оперативной памяти. Такая база данных очень мала по объему и ограничивает использование языка. Было бы полезным взять хороший пакет программ управления базой данных и разработать для него интерфейс конечного пользователя на основе языка Пролог.

В этом случае Вы можете обращаться, скажем, к базе данных о работниках кино с различными вопросами, связанными с изучением их личных контактов, деловых связей и т. д.

Вы можете составить такой запрос: найти режиссеров, которые работали, по крайней мере, над двумя фильмами вместе с режиссерами, имеющими опыт создания фильмов с прибылью более 50 млн. фунтов стерлингов и никогда не создававшими убыточные картины.

Таким образом, Вы можете собрать вместе очень компетентных специалистов. Ассоциативным путем Вы можете даже доказать виновность подсудимого, хотя судьи всегда решительно возражали против такого подхода.

ГЛАВА 6
ИСТОЧНИКИ И СТОИМОСТЬ ИНФОРМАЦИИ

До недавнего времени наиболее распространенными пакетами программ для микроЭВМ были пакеты обработки текстов и расчета рабочих таблиц. Они используют небольшие файлы и потеря информации в случае ошибки оператора или сбоя ЭВМ не приводила к ощутимому ущербу. Конечно, неприятно потерять главу из книги или результаты финансирования расчетов, но дело это поправимое, чего не скажешь об информации, хранящейся в базе данных. Стоимость данных в базе может во много раз превосходить стоимость ее программных и технических средств, а потеря данных вообще может оказаться невосполнимой и привести предприятие к краху.

Сколько же стоят данные? Пока этот вопрос волнует немногих, главным образом фирмы, продающие информацию. Но он будет все чаще и чаще возникать у владельцев персональных ЭВМ в будущем. Существует несколько элементов затрат . Во-первых, данные надо собрать. Они должны быть зафиксированы в каком-либо виде на бумаге, прежде чем их можно будет вводить в ЭВМ. Это могут быть результаты какого-либо единовременного обследования, анкетных опросов, отзывы на рекламную кампанию. Вряд ли можно дешевле, чем за 10 пенсов, зафиксировать какие-либо сведения на бумаге. Затраты на сборы данных могут быть во много раз большими. Примем за максимум 50 пенсов и 25 пенсов как разумное среднее. Затем данные надо набрать на клавиатуре. Квалифицированная машинистка делает до 300 ударов в минуту, а запись длиной в 300 символов – это вполне приемлемая для базы данных запись.

Оплата труда машинистки вместе с накладными расходами составляет 10 000 фунтов стерлингов в год или 10 пенсов в минуту. Аналогичные затраты связаны с контролем данных (см. далее о необходимости такого контроля). Затем надо учесть капитальные затраты и эксплуатационные расходы на ЭВМ, в памяти которой хранятся данные. МикроЭВМ, пригодная для серьезной работы с базой данных, стоит не менее 4000 фунтов стерлингов. В состав ЭВМ входит накопитель на твердом диске емкостью 15М байт и печатающее устройство. С учетом процентов за кредит, стоимости технического обслуживания и ремонта за 5 лет, амортизационных отчислений затраты на оборудование составят 1800 фунтов в год. Самая маленькая база данных, которую экономически оправданно вести на микроЭВМ, занимает 1М байт дисковой памяти (3000 на 300 байт). Стоимость сбора данных для такой базы – 1350 фунтов (3000 X 45 пенсов). Каждый год изменения будут вноситься в 20% записей (эта цифра колеблется в зависимости от приложения), поэтому расходы на поддержание базы данных составят еще 270 фунтов в год.

Таким образом, даже самая маленькая база данных будет стоить 3150 фунтов в первый год и 2070 фунтов каждый последующий. И это не считая заработной платы с накладными расходами руководителя проекта. А если добавить его или ее поседевшие волосы, то сумму можно удвоить.

У пользователей появляется желание иметь все большие и большие базы данных на микроЭВМ. Так, некоторые члены британского парламента хотели бы хранить в машине данные о своих избирателях. Когда подойдут выборы, они смогут направить избирателям специальные открытки. Член парламента хотел бы лично написать работникам организаций, связанных с охраной окружающей среды, и сообщить о великолепных результатах голосования по вопросу об охране тюленей или защиты лесных массивов.

Подобные затеи выглядят очень заманчиво, но на практике обходятся слишком дорого. Избирательный округ насчитывает в среднем 60 000 избирателей. Их имена и адреса есть в домовых книгах (есть они и на магнитной ленте, но министерство внутренних дел отказывается передать их кому-либо, см. § 6.4 о защите информации). Эту информацию надо ввести через дисплей. 60 000 записей в базе данных (принимаем опять, что длина записи 300 байт) требует 18М байт памяти. Индексные файлы займут еще 18М байт. Для такой базы данных нужна весьма солидная микроЭВМ, хотя бы потому, что заполнение и ее ведение требуют усилий нескольких человек, поэтому к процессору должно быть подключено несколько терминалов. Машина такого класса будет стоить примерно 16 000 фунтов, ежегодные приведенные капитальные затраты (подсчитанные по той же схеме, что и раньше) составят 7200 фунтов. Стоимость информации составит не менее 27 000 фунтов (списки избирателей Вы получите бесплатно, но сбор данных для решения других задач требует затрат). Еще 5400 фунтов в год надо заплатить за поддержание базы данных. Вся затея начинает выглядеть очень дорогой. Даже если такие затраты оправданы, важно иметь в виду, что любой полезный массив данных может стоить дороже, чем стоимость программного и технического обеспечения вместе взятых. Поэтому неразумно пытаться сэкономить деньги на технике или программах, использование которых может поставить под угрозу целостность Ваших данных.

Если есть возможность, то лучше заплатить за готовую информацию, записанную на машинный носитель, чем вводить ее в ЭВМ. Конечно, не все базы данных продаются. Никто не продаст Вам сведения о клиентах фирмы или ее служащих и состоянии их банковских счетов. Но данные о потенциальных покупателях натяжных гаечных ключей безусловно могут быть представлены за плату фирме, изготовляющей такие ключи. Этими данными располагают специалисты по сбору подобной информации. Они имеют базы данных по всей номенклатуре изделий, получаемых предприятиями в данной отрасли или на данной территории и могут извлечь из них сведения, представляющие интерес для фирмы – изготовителя натяжных гаечных ключей. Такие услуги выполняются уже давно, правда, в менее совершенном виде: по почте высылаются списки, содержащие необходимую информацию. Вы можете, если в этом есть необходимость, приобрести списки лиц с самыми необычными интересами. Хотите знать, кто из учителей планирует поехать в отпуск в туристическую поездку? Пожа-

луйста. Вас интересуют водолазные компании, которые покупают водород? Пожалуйста.

В прошлом Вы покупали такой список и отправляли по указанным в нем адресам свои проспекты в надежде установить достаточное число контактов и оправдать всю затею. Гораздо лучше, если Вы создадите свою собственную базу данных. Начните с приобретения списка потребителей натяжных гаечных ключей, записанного на гибком магнитном диске. Информацию с диска считываете в Вашу базу данных. Теперь она содержит сведения о Ваших потенциальных покупателях. С помощью микроЭВМ подготавливаете и рассылаете им письма. Затем вводите в базу данных их ответы. Какая-то компания переехала, другая хотела бы связаться с Вашим представителем, третья заинтересована в настоящее время не столько в гаечных ключах, сколько в других приспособлениях и спрашивает, можете ли Вы помочь.

Спустя некоторое время Вы добавляете к первоначальной информации множество новых полезных данных и исходная база изменится до неузнаваемости. Как только Ваши изделия начнут покупать, список потенциальных покупателей превращается в список клиентов. Вы начнете добавлять записи о торговых операциях, накапливать данные для программ, решающих задачи бухгалтерского учета, которые также будут обращаться к базе данных. Пока мы исходили из того, что все данные вводятся в микроЭВМ самими пользователями.

Современный этап компьютеризации чем-то похож на начальный период книгопечатания, когда основная проблема заключалась в том, чтобы получить оттиск на бумаге. Когда был впервые основан печатный цех, вряд ли кто думал писать книги для издания. Не было профессиональных писателей, литературных посредников, редакторов и издателей. Все, что имело смысл читать, шло в набор, а затем находились один–два человека, которые покупали печатную продукцию. Четыре столетия спустя ситуация в издательском деле изменилась. Сейчас намного проще напечатать что-либо, чем найти что-то, достойное публикации.

Несомненно, аналогичные перемены должны произойти в применении вычислительной техники. Пока мы продаем людям микроЭВМ и говорим: "С помощью ЭВМ вы можете создать собственный словарь, телефонный справочник, налоговый справочник и сотни других полезных книг. Желаем успеха!!!". Некоторые из нас могут создать такие книги, если очень постараются и найдут для этого время, но будет гораздо проще, если эту работу сделает кто-то другой. Не только проще, но и дешевле, поскольку затраты будут распределены на большое число покупателей. То же самое можно сказать по отношению к программному обеспечению и информации для ЭВМ.

6.1. РАСПРЕДЕЛЕНИЕ ИНФОРМАЦИИ

Если продолжить аналогию между базами данных и книгами, то неизбежно возникает вопрос о распределении информации, содержащейся в базах данных, на коммерческой основе. Книга не имеет коммерческой ценности, другими словами не будет издана, если ее нельзя будет распространить среди достаточного числа читателей и получить с них деньги. Наряду с прогрессом, достигнутым за 400 лет в технологии издательского дела, не меньший прогресс был достигнут в организации книжного рынка. Сейчас мы имеем хорошо отработанные каналы для сбыта печатной продукции. Потребности организованного книжного рынка формируются читательской массой. В англоговорящих странах читатели хорошо представляют, о чем должна быть книга, как она должна выглядеть, сколько стоить. Книжные магазины связаны с книжными оптовыми базами, а те в свою очередь имеют дело с издателями. Последние располагают денежными средствами, которые они вкладывают в новые книги. Они знают, какую прибыль могут принести те или иные книги. Издатели нанимают персонал, который осуществляет издание, редактирование, вычитку книг. И, наконец, есть немалое число образованных людей, располагающих информацией, которая стоит того, чтобы ее отразили в книгах. Они выступают в качестве авторов.

Ничего похожего нет пока на рынке микрокомпьютеров. Этот рынок только начинает формироваться. Большие объемы информации хранятся на больших ЭВМ, но каналы, по которым эта информация может распространяться, еще весьма ограничены. В мире насчитывается примерно 1500 баз данных, хранящихся в больших машинах, к которым за определенную плату могут обращаться специалисты. Например, базы данных компании "Рейтер" об операциях на фондовой бирже, фирмы "Лексис" о законодательных актах, фирмы "Дейтасолвес" о международных событиях и др. Чтобы получить доступ к такой базе данных, необходим телефонный канал связи, микроЭВМ, модем (см. § 1.8) и достаточное терпение, поскольку все они работают по-разному.

Большим недостатком таких баз данных является то, что они повторяют традиционные системы информации. Вы вводите запрос, ЭВМ осуществляет по нему поиск и выдает интересующие Вас данные. Данные отображаются на экране Вашего терминала, Вы их читаете и не больше. Вы не можете переписать, создать или пополнить на ее основе свою базу данных или продать информацию, которой Вы располагаете и которая могла бы заинтересовать пользователей баз данных на больших ЭВМ.

Одно из очевидных решений этой проблемы заключается в том, чтобы продавать магнитные диски с записанными на них базами данных. С этой информацией должны работать пакеты программ управления базами данных, имеющиеся у пользователей. Программное обеспечение должно позволять расширение записей в базе. Тогда появляется возможность добавлять собственные данные к готовой базе данных и даже продавать новую

информацию. Одним из факторов, сдерживающих такое распространение информации для микроЭВМ, является отсутствие единого стандарта на формат гибкого диска. Теоретически диски фирмы ИБМ диаметром 8 дюймов (203 мм) с записью на одной стороне одинарной плотности должны быть совместимы со всеми ЭВМ, которые имеют в своем составе такие накопители. На практике это достигается далеко не всегда. К сожалению, большинство фирм – изготовителей микроЭВМ отдают предпочтение дискам диаметром 5 1/4 дюйма (133 мм) и устанавливают для них собственный стандарт записи. В настоящее время существует более 200 форматов записи на диски диаметром 5 1/4 дюйма. Это оборачивается настоящим кошмаром для фирм, разрабатывающих программное обеспечение, и делает невозможным распространение баз данных на гибких дисках, корректировку которых требуется осуществлять ежедневно. Внедрение баз данных на микроЭВМ связывается также с широким применением твердых дисков, на которых можно хранить базы данных достаточных размеров. Решение обеих проблем упрощается с выпуском в качестве практического стандарта персонального компьютера *IBM PC*. Большинство шестнадцатиразрядных ЭВМ, работающих в операционной среде *MS-DOS* или *CP/M*86, будут теперь читать диски в формате *PC*. "Обозреватели отрасли" (люди, которые всегда говорят то, что думают другие люди, их цитирующие) предсказывали, что к 1986 г. в США будет 9млн. профессиональных микроЭВМ и 1 млн. в Англии. Начинает складываться серьезный рынок сбыта микроЭВМ. Многие машины такого класса (включая, безусловно, ЭВМ *IBM XT*) имеют в своем составе твердые диски и их стоимость постоянно снижается. Созрели условия для создания отрасли "микроинформатики". Хотя сейчас уже существуют проекты создания информационного общества, в котором к каждому дому будут проведены световоды большой мощности, магнитный диск, переданный по почте, служит быстрым и удобным способом обмена информацией. Другая возможная альтернатива – передача данных по телефонным каналам. По телефонным кабелям общего назначения данные могут передаваться со скоростью до 1500 бит/с. Обычная скорость передачи данных по телефонным каналам (она же и максимальная, если на одном конце используется дешевый модем с акустической связью) составляет 300 бит/с. Для передачи реальной базы данных, ориентированной на экономические приложения объемом 5М байт, потребуется не менее 37 ч. Будет весьма удивительно, если данные удастся передать без искажений с первой попытки. Поэтому фактическое время передачи может составить неделю и более. По почте Вы получите данные быстрее.

Однако можно найти промежуточное решение между личной базой данных на настольной микроЭВМ, установленной на рабочем месте пользователя, и базой данных коллективного пользования на большой ЭВМ, выдающей информацию по запросу. Работы, которые ведутся в этом направлении и, безусловно, будут развиваться, связаны с реализацией концепции электронной почты. Система электронной почты использует большую ЭВМ с выходом на телефонную сеть. Примером может служить си-

стема "Бритиш Телеком Голд", связанная напрямую с сетью "Диалком" американской фирмы ИТТ и 18 другими аналогичными сетями в других странах. Каждый абонент имеет почтовый ящик с номером, например 81 : *ABC* 123. Первые две цифры показывают, с какой ЭВМ выходит на связь пользователь. В нашем примере 81 означает одну из двух машин, обслуживающих Лондон. Три буквы идентифицируют пользователя, а три цифры обозначают один из многих используемых почтовых ящиков. Для работы с сетью Вы набираете специальный телефонный номер напрямую или, если Вы живете за пределами местной телефонной сети, набираете номер сети с коммутируемыми пакетами сообщений и через нее получаете по низкому тарифу доступ к электронной почте.

Во многие модели микроЭВМ в настоящее время встраиваются модемы с автоматическими номеронабирателями. Соответствующее программное обеспечение позволяет устанавливать связь и подключаться к большой ЭВМ автоматически.

По существу, почтовый ящик каждого абонента представляет собой запись в базе данных. База распределена по многим различным ЭВМ, но специальный комплекс программ позволяет все соединения между ЭВМ выполнять автоматически. Абоненты могут посылать друг другу сообщения и вести диалог примерно так же, как это делается по телетайпу, но

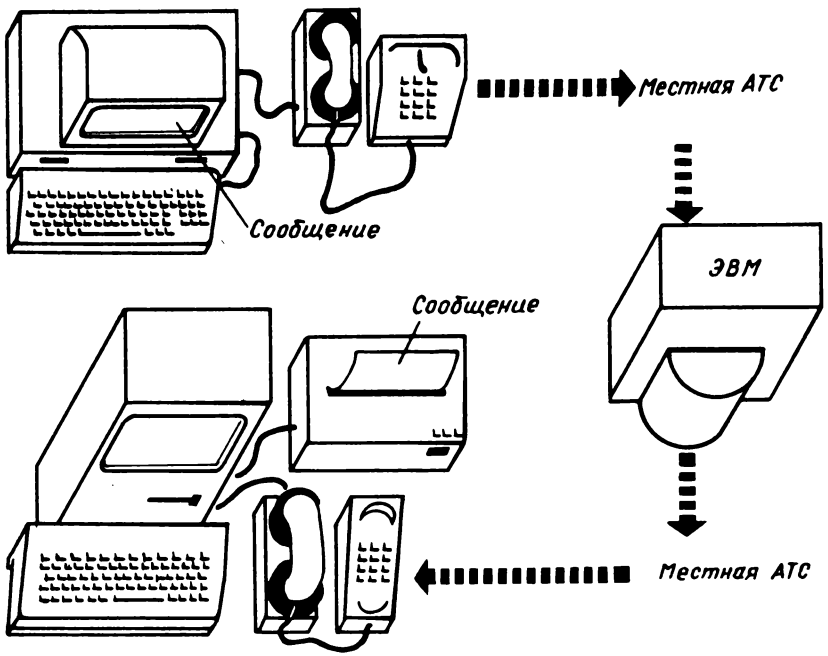


Рис. 42. В системе электронной почты абоненты используют свои компьютеры для посылки и получения сообщений из базы данных большой ЭВМ

в более естественной форме. Если группа абонентов имеет общий почтовый ящик, то члены группы могут проводить электронные "совещания", посылая друг другу сообщения по вопросу повестки дня. Вы подключаетесь к сети, читаете свою почту, просматриваете самые последние замечания и предложения по обсуждаемому вопросу, еще раз просматриваете высказывание Билла и пишете свои собственные замечания, в которых указываете на ошибочность идеи Аллена и в то же время привлекаете внимание к тому, что сказал Джим три недели назад, и что теперь стало очевидным. Работники одного и того же отделения фирмы могут посредством электронной почты непринужденно вести между собой переговоры, не ощущая даже, что их разделяют многие тысячи километров.

Премущество системы электронной почты, основанной на центральной ЭВМ, заключается в том, что в качестве терминального устройства может применяться не только дисплей, но и персональная ЭВМ, через которую пользователь обращается к системе с запросом. Запрос передается базой данных электронной почты другой ЭВМ, на которой хранится своя база данных. Таким образом, электронная почта служит отличным средством распределения данных. Например, пользователь базы данных посылает запрос: "Пожалуйста, выдайте список всех предприятий Монитобы, которым требуются разводные гаечные ключи и на которых работает более 50 человек". Система электронной почты передает его в соответствующую базу данных (появятся базы данных о базах данных), снимает со счета абонента плату по тарифу за информационную услугу и помещает интересующую абонента информацию в его почтовый ящик. Эта информация может быть загружена в собственную базу данных абонента и, при необходимости, обработана в соответствии с программой пользователя.

6.2. ОБРАБОТКА ИНФОРМАЦИИ

Довольно трудно предвидеть, какие формы могут принять базы данных в будущем. На первый взгляд обычные справочники, которыми Вы пользуетесь на рабочем месте: телефонные книги, каталоги, словари, торговые справочники, т. е. все, что имеет вид перечня,должно быть заведено в базу данных. Но,с другой стороны,развитие баз данных может пойти по иному пути, хотя основываться они будут на той же информации. Интересно проследить эту мысль на примере телефонной книги. В этом справочнике, по существу, собраны сведения о всех экономически и социально активных гражданах, проживающих в данном районе. Есть их фамилии, номера телефонов, адреса и почтовые индексы. Это полезная информация, но есть и другая, не менее полезная информация, которую можно было бы добавить к первой. Когда мы устанавливаем деловые контакты с новыми людьми, нам хотелось бы узнать, что они делают и сколько примерно зарабатывают (в США Вы просто спрашиваете об этом напрямую, в Англии надо потратить огромное количество времени на то, чтобы узнать об этом

2. Все население

Возраст	Численность населения	Мужчины		Женщины		Пенсионеры мужчины
		одинокие	семейные	одинокие	семейные	
0–4	57	59	xxx	62	xxx	xxx
5–9	64	66	xxx	69	xxx	xxx
10–14	71	73	xxx	76	xxx	xxx
15	78	80	xxx	83	xxx	xxx
16–19	85	87	88	90	91	xxx
20–24	93	94	95	97	98	xxx
25–29	99	101	102	104	105	xxx
30–34	106	108	109	111	112	xxx
35–39	113	115	116	118	119	199
40–44	120	122	123	125	126	200
45–49	127	129	130	132	133	201
50–54	134	136	137	139	140	202
55–59	141	143	144	146	147	203
60–64	148	150	151	153	154	204
65–69	155	157	158	160	161	205
70–74	162	164	165	167	168	206
75–79	169	171	172	174	175	207
80–84	176	178	179	181	182	208
85	183	185	186	188	189	209

5. Все население в возрасте от 16 лет и более

Трудовая активность населения	Численность населения	Мужчины		Женщины	
		одинокие	семейные	одинокие	семейные
Все лица в возрасте 16+	380	382	383	385	386
Лица, занятые трудовой деятельностью	387	389	390	392	393
Работающие	394	396	397	399	400
Безработные	401	403	404	406	407
Временно утратившие трудоспособность	408	410	411	413	414
Лица, не занятые трудовой деятельностью	415	417	418	420	421
Хронические больные	422	424	425	427	428
Пенсионеры	429	431	432	434	435
Учащиеся	436	438	439	441	442
Прочие	433	445	446	448	449

Рис. 43. Фрагмент многотомного статистического обследования. Приведены данные по определенной территории

и не показаться невежливым). Эта информация – часть ее – содержится в статистических обследованиях, проводимых один раз в 5 лет. Органы статистики собирают данные о размерах личных доходов. В Англии, чтобы сохранить секретность, сведения о доходах выдаются только из расчета на 100 семей. Но люди с одинаковым уровнем доходов стараются держаться вместе, поэтому эти данные могут быть весьма полезными.

К сожалению, подобная база данных получается очень больших размеров. Органы статистики используют в базе около 100 000 записей. Каждая запись состоит примерно из 4000 трехзначных цифровых полей. Это составит 1,2Г байт. Практически реализовать базу данных такого объема можно будет только после серийного освоения промышленностью оптических дисков (см. гл. 10). Однако Вы можете сгруппировать эту информацию по 10 000 районов с численностью населения в каждом около 5000 человек или с числом семей примерно 1000. Многие специалисты по изучению конъюнктуры рынка и сбыту были бы рады иметь такую базу данных на своей микроЭВМ. Как заявил руководитель одной из крупных компаний: "Если Вы правильно учитываете торговые издержки и включаете в них стоимость нормативных запасов, расходы на управление, содержание складов, движение товаров, то их доля в себестоимости составит 40 %. Даже небольшие улучшения в этой области дадут экономический эффект. База данных может стать важным инструментом совершенствования".

База данных, основанная на результатах статистических обследований, может быть значительно расширена. Помимо сведений о личных доходах, жилищных условиях, количестве автомобилей, находящихся в индивидуальном пользовании, важно знать, что производится на этой территории, какие имеются предприятия, какие здесь работают коммерческие каналы телевидения, какие используются почтовые индексы и т. д.

Наш старый знакомый – изготовитель разводных гаечных ключей – начал создавать базу данных, имея на руках лишь список потенциальных покупателей своей продукции. Почему бы к этому списку не добавить различные данные о каждом предприятии, каждом клиенте, включая сведения о квалификации, профессиональных интересах, хобби. В социальном плане такая база отражает экономическое положение каждого человека в округе: где он живет, чем занимается и где, как у него идут дела в данный момент. Если человек ищет другую работу, он может воспользоваться информацией о вакансиях, которые также хранятся в базе данных.

База данных может строиться на принципах электронной почты и тогда Вы получаете возможность обмениваться сообщениями. Она поможет Вам установить, кто чем увлекается в свободное время и если у Вас есть интересное предложение, скажем, для филателистов, Вы можете быстро оповестить о нем всех коллекционеров. База данных поможет найти заинтересованного покупателя, если Вы захотите что-либо продать. Система может выполнять функции банка: Вы делаете вклад и получаете право пользоваться кредитом и совершать электронные сделки.

Большая часть этой обширной информации может быть введена в базу

данных только по инициативе самого абонента, который добровольно сообщает сведения о себе, поскольку его интересуют такие же сведения о других. В 2000 г. Вы можете вести полнокровную общественную и профессиональную жизнь только при условии, что запись о Вас в "телефонной книге" очень обстоятельна и достоверна. Если Вы устали от общения с людьми и хотите уединиться, Вы просто удаляете запись о себе из базы данных. Если Вам надоели филателисты, Вы вводите защитный код для этой части записи и сообщаете о нем своим друзьям. Теперь только друзья могут прочесть сведения о Вашей страсти к коллекционированию марок. Защита информации становится очень важной проблемой. Есть сведения, которые Вы не скрываете от друзей, но не хотели бы сообщать всем подряд. Системы управления базами данных должны осуществить целую иерархию сложных проверок. Они также должны подтверждать некоторые данные. Например, если кто-то заявляет, что он по профессии доктор, база данных должна подтвердить это.

Необходимо будет найти электронный эквивалент подписи на документах, чтобы удостоверить их подлинность. Нет пределов возможному уровню детализации данных. Сейчас мы вынуждены оперировать единицами измерения в миллионах, проводя грубые статистические оценки. Мы избираем правительство на 5 лет, исходя из того, что большинство избирателей думает об обещаниях, которые им раздают накануне выборов. Когда базы данных будут широко внедрены, можно будет провести мгновенный референдум по любому вопросу. Ваше мнение по любой текущей проблеме добавляется к Вашей записи. Это нежелательно, так как избиратели голосуют скорее не за комплекс мер и мероприятий, а за конкретного человека, который,на их взгляд,поведет себя разумно в непредвиденных обстоятельствах, в которых может оказаться правительство. Поскольку избиратель не имеет возможности глубоко вникнуть в проблемы, то "электронная демократия" может стать настоящим бедствием. Это равносильно ситуации, когда каждый пассажир автобуса может крутить свое рулевое колесо, а ЭВМ суммирует углы поворота и рассчитывает некий средний угол, который и определит дальнейшее направление движения автобуса.

Очень интересные результаты можно получить, не дожидаясь такой высокой степени автоматизации. Сегодня к распределению и использованию ресурсов мы подходим с весьма общих позиций. Если Вам надо совершить перелет на противоположный берег Атлантического океана, то сначала Вы выбираете авиакомпанию, затем смотрите, есть ли рейс в подходящее для Вас время, а затем, когда рейс выбран, узнаете, есть ли на этот рейс свободное место. Что пассажиру действительно нужно, так это место в салоне авиалайнера, летящего в Нью-Йорк. Имеет значение также время рейса и стоимость билета. В большинстве случаев неважно, какая авиакомпания осуществляет данный рейс, все они обслуживают пассажиров одинаково плохо. Гораздо лучше иметь базу данных об авиарейсах, доступ к которой имеет каждый. Сначала мы получаем ответ, есть ли свободное место на интересующем нас рейсе, а затем уже становится извест-

ной авиакомпания, выполняющая данный рейс. Если бы объединить такую базу данных с системой продажи билетов, то весь кошмар резервирования мест на самолет просто перестал бы существовать. Одновременно высвободилось бы громадное число служащих, поскольку всю необходимую работу выполняли бы пользователи на своих микроЭВМ. Аналогичным образом можно продавать все виды товаров: дома́, автомобили, даже пудру в мелкой упаковке. Как только в большинстве учреждений и жилых домов появятся телекоммуникационные устройства с высокой разрешающей способностью, мы получим возможность рядом с текстовой информацией помещать графическое изображение высокого качества.

6.3. ДОСТОВЕРНОСТЬ ИНФОРМАЦИИ

До сих пор мы говорили об информации, хранящейся в базе данных, не задаваясь вопросом, как обеспечить безошибочность и точность информации. В реальных условиях не всегда удается это сделать. В книжном деле функция редактирования необходима, хотя она и мало заметна. Редакторы стараются сделать так, чтобы книга была написана литературным языком и соответствовала неким подсознательным нормам, с которыми читатели подходят к книгам различного жанра. Например, Вы будете удивлены, если в развлекательном романе встретите ссылки в конце страницы или увидите в справочнике по международной торговле джутом, абзац, который начинается словами "Была темная ненастная ночь ...".

Задача редактора – контролировать, что выходит в печать от имени издательства. Торговый знак издательства служит гарантией качества книги. Одна из причин, по которой люди вкладывают столько усилий, чтобы издать книгу по всем правилам, заключается в том, что издание требует очень больших затрат. Самая дешевая книга обходится издательству в 10–20 тысяч фунтов стерлингов. Стоимость бумаги, типографского набора, торговые издержки составляют большую часть затрат. При издании баз данных эти статьи расходов, безусловно, отсутствуют, но потребители должны приобрести ЭВМ и заплатить за телефонный канал. В принципе это хорошо, так как повышается конкурентоспособность небольших самостоятельных организаций. Однако на практике затраты на проверку данных оказываются столь высокими, что могут сравниться с затратами на первоначальный сбор данных. Они начинают играть важную роль в стоимости базы данных, в то время как при печатном издании информации они составляют лишь небольшой процент.

Появляется соблазн – избежать дорогостоящей операции верификации данных наподобие того, как небольшие авиакомпании стараются избежать необходимой, но малозаметной работы по техническому обслуживанию своих самолетов. Как сказал один рабочий судоверфи другому, объясняя, почему он не заделал дырку в корпусе корабля: "Ее никто не увидит, она же под водой".

Ошибки в данных или книгах неприятны тем, что они остаются незамеченными до тех пор, пока не подведут Вас, если только Вы не позабо-

титесь выявить их раньше. Сколько раз мне приходилось слышать от программистов: "В этой программе ошибок больше нет!". А через неделю (или еще хуже, через полгода) он заявлял: "Не понимаю, как эта программа вообще работала!". То же самое можно сказать о данных. Организации – держатели баз данных должны будут затрачивать на выверку данных большие средства. До тех пор пока они не смогут гарантировать отсутствие ошибки, покупатели будут относится к готовым базам данных подозрительно. Вообразите, например, какие результаты Вы можете получить при расчете на ЭВМ капитальных вложений, если данные о ценах на оборудование ошибочны.

Обеспечение достоверности информации станет серьезной проблемой в издании баз данных. С этой проблемой уже столкнулись в некоторых американских университетах, где каждый студент имеет доступ к вычислительным системам. Курсовые задания, как правило, составляются преподавательским составом и записываются на машинные носители. Неудивительно, что текст содержит множество ошибок. Большую часть из них можно было выявить, если бы текст прошел минимальную редакционную обработку, как это делалось в докомпьютерную эпоху. Аналогичная ситуация сложилась в Англии с пакетами для обучения математике. Информация, получаемая из ЭВМ, всегда выглядит правильной. В программное обеспечение микроЭВМ, как правило, включается пакет программ расчета рабочих таблиц. Я часто использую этот пакет для подготовки финансовых отчетов о деятельности предприятия. Недавно ЭВМ выдала на печать очень хорошие показатели. Руководители соответствующих служб были в восторге, пока я не заметил, что результаты ошибочны. Подсчитывая итоги по столбцу, программа суммировала только часть строк. Достаточно было чуть внимательнее посмотреть на цифры, чтобы ошибка бросилась в глаза, но никто этого не сделал – ведь цифры на распечатке выглядели такими правильными!

Зарплата	6000
Аренда	2000
Реклама	26000
Телефон	500
Всего	8000

Программа сложила только первые две цифры. Вероятно, она оказалась испорченной на более ранней стадии расчетов. Ошибка осталась незамеченной, так как весь смысл применения ЭВМ в том и заключается, чтобы освободить человека от расчетов, и, конечно, никому не пришло в голову сложить пару цифр в уме.

Надо исходить из того, что все результаты, получаемые на ЭВМ, неверны, пока не будет доказано обратное. И единственным убедительным доказательством является длительное использование ЭВМ. Если данные или программа долгие годы использовались многими людьми и не вызы-

вали нареканий, можно считать, что они правильные. Безусловно, основная предпосылка при работе с базами данных в диалоговом режиме заключается в том, что хранящаяся информация всегда самая новая – и поэтому, почти по определению, ненадежная.

6.4. ЗАКОНОДАТЕЛЬСТВО О ЗАЩИТЕ ИНФОРМАЦИИ

Еще одним препятствием на пути широкого распространения баз данных может стать будущее законодательство о защите данных. Нормативные акты, разрабатываемые сейчас в Великобритании, преследуют следующие цели:

1) предоставить людям возможность убедиться в том, что информация о них, хранящаяся в памяти ЭВМ, правильная;

2) если информация закрытая, например медицинские записи, то доступ к ней могут иметь только те, кто должен об этом знать.

Предусматривается, что держатели баз данных должны иметь на это специальное разрешение и любой человек может просмотреть касающуюся его запись и добиваться внесения изменений, если она неверная. Неясно, почему такой подход должен применяться только к электронным хранилищам информации и не распространяется на печатные картотеки, в которых и сведений часто хранится больше, и последствия ошибок могут быть серьезнее. Трудно также представить себе, как будет применяться этот закон, когда на каждом из многих миллионов персональных компьютеров может храниться база данных. К 1990 г. распечатка перечня баз данных, в которых содержатся личные данные, будет иметь массу 500 кг. Чтобы просмотреть ее, потребуется вся жизнь.

Что касается ограничения доступа к информации, то эта проблема либо решается частично, либо используются совершенно не поддающиеся дешифровке коды, которые слишком сложно будет применить.

6.5. ЗАБЫТОЕ ПРОШЛОЕ

Еще одна проблема связана с оперативной корректировкой информации, хранящейся в базах данных. Меняется информация, и все заинтересованные пользователи немедленно получают скорректированный вариант базы данных. Старый вариант навсегда исчезает. Большим преимуществом печатных изданий было то, что все издавалось в огромных количествах. После того как набор был сделан, экономически выгодно было получать большое число копий, так как стоят они очень дешево. В результате практически все, что было когда-либо напечатано, сохранилось. Многие библиотеки в мире накапливают ежедневные газетные издания. Хотя бы один экземпляр каждой новой книги посылается в центральные библиотеки страны. Помимо этих обязательных мер по сохранению изданий, все, что публикуется большими тиражами, оседает в ящиках столов, на полках периферийных библиотек, на чердаках и в чемоданах граждан и спустя долгое время (иногда через столетия) может быть найдено.

Для многих религиозных организаций, например католической церкви,

изобретение печатного станка имело двоякие последствия. С одной стороны, это изобретение позволило верующим широко распространять свои убеждения. Но оно давало возможность еретикам также легко распространять свои взгляды на мир. За прошедшие столетия было опубликовано много книг, которые Ватикан хотел бы уничтожить. Однако только в отдельных случаях церкви удавалось собрать и передать палачу для сожжения весь тираж. Но это были книги, изданные на редких языках (например, тосканском), у которых не было читателей вне территории, контролируемой римской католической церковью. Антирелигиозные книги, изданные на латинском или даже на итальянском языке, быстро распространялись по всем европейским странам и оказывались недосягаемыми для церкви. Специалисты, изучающие историю печатного дела, отыскали, по крайней мере, по одному экземпляру всех книг, изданных на Западе за последние пятьсот лет.

Даже государственные секреты должны быть на чем-то записаны и где-то храниться – нередко в нескольких местах и различными людьми. И как результат: почти все политические тайны, существовавшие в истории Европы за последние пять веков, оказались разгаданными. Не плохо было бы продолжить эту традицию на следующие пятьсот лет. С переходом к электронным хранилищам мы в каждый конкретный момент времени будем иметь только один экземпляр базы данных – самый последний, скорректированный. Архивизация данных не происходит естественным путем, ее надо осуществлять преднамеренно. А все, что делается преднамеренно, часто делается плохо.

Содержание библиотек, их книжных фондов обходится очень дорого, но до недавнего времени им не было замены. Любое учебное заведение – даже такое скромное, как начальная школа – должно иметь библиотеку. Для университета содержание библиотеки обходится в миллион фунтов стерлингов в год.

Но придет эра электроники и библиотеки станут ненужной роскошью, люди разучатся хранить книги. Негде будет хранить даже те клочки бумаги, которые переживут электронный ураган. История исчезнет. Кто-то справедливо заметил, что нация, у которой нет истории, не имеет будущего. История – великий учитель. Она учит, что в мире действуют свои собственные законы, которые нельзя нарушать.

Первое, что делают диктаторы, приходя к власти, – это пытаются переписать историю. Люди надежно защищены от ошибок, если они знают, что в действительности произошло. И хотя электроника – это самое современное направление технического прогресса, не следует думать, что технический прогресс избавляет человечество от таких форм правления, как диктатура. Ничто так не радует диктатора, как невозможность найти в прошлом опровержение того, что он сейчас делает.

ГЛАВА 7
БАЗЫ ДАННЫХ, ОРИЕНТИРОВАННЫЕ НА ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Термин "искусственный интеллект" (сокращенно ИИ), безусловно, привлекал внимание каждого, кто хотя бы поверхностно интересуется вычислительной техникой. Исследования по ИИ выделились в самостоятельное научное направление, приверженцы которого обещают получать в будущем замечательные результаты. У нового направления есть и противники. Они считают его таким же бессодержательным и псевдонаучным, как френология. Тем не менее, если не воспринимать ИИ слишком серьезно, можно реализовать интересные приложения, используя программы ИИ для работы с базами данных.

Существенное различие между обычным программированием и программированием ИИ заключается в том, что в первом случае программисты знают, что они хотят от ЭВМ в мельчайших деталях. Трудность состоит в том, чтобы заставить ЭВМ сделать то, что они хотят. При разработке программ ИИ программисты не знают, что должен делать компьютер. Но они представляют себе, как можно заставить ЭВМ узнать, что же ей делать. Программисты воплощают свои идеи в программе и с интересом наблюдают за результатами работы ЭВМ.

Специалисты, работающие в области ИИ, говорят с улыбкой: "Если Вы понимаете, как это работает, это не ИИ". Есть и более короткий вариант этой фразы: "Если это работает, это не ИИ".

Каким образом программа ИИ может взаимодействовать с программами управления базой данных? Как такая "интеллектуальная" база данных работает?

Пока самым большим достижением в области ИИ можно считать "экспертные системы". Интеллектуальная база данных выполняет функции, обратные функциям экспертной системы. Необходимо пояснить, как работает экспертная система. Эксперты на основе собственного опыта или глубокого изучения проблемы осмысливают большое число фактов и устанавливают правила, их объясняющие. Выработанные ими правила упорядочиваются и хранятся в памяти ЭВМ. К ЭВМ могут обращаться за консультацией другие специалисты.

Например, экспертная система по ликвидации аварий на буровых вышках будет содержать информацию, которую соберут люди, хорошо знающие работу установок. Эксперты (будем надеется, что это так) рассмотрят все возможные неприятности, которые могут обрушиться на руководителя буровых работ. Они постараются ответить на вопросы: что должен делать руководитель, если загорятся на буровой топливные баки для вертолета? Что необходимо предпринять, если откажет клапан, установленный в скважине? Как быть, если одна из опор вышки прогнулась? Одна авария может повлечь за собой многие другие аварии, а у руководителя нет времени учесть все обстоятельства в минуту

опасности. Используя ЭВМ и экспертную систему, руководитель может сообщить ей, что случилось и получить конструктивный совет.

На первый взгляд экспертная система создает не более чем указатели к определенной области знаний. Если у Вас есть справочник по авариям на буровых установках, Вы можете просмотреть содержание и найти признаки угрожающей Вам катастрофы:

. . .
Пожары стр. 45
Пожары, склад горючего для вертолета стр. 57
и т. д.

Возможности экспертной системы шире. Подобно базе данных, она позволяет проводить поиск по ряду признаков одновременно. Например: пожар, топливо, вертолет. Система попросит Вас оценить вероятность событий, описание которых Вы используете в качестве условий поиска. Так же как и руководитель буровых работ, пробудившийся от беспокойного сна и с недоумением созерцающий красные отблески на иллюминаторе, Вы можете не знать, что горит. И на запрос системы: "Насколько Вы уверены в том, что горят топливные баки для вертолета?", Вы ответите: "50 %". Такие системы используются на практике, например при определении диагноза заболевания.

"Насколько Вы уверены в том, что больной пил в течение месяца зараженную воду?" Врач может не знать этого, а больной может находиться в тяжелом состоянии. И тогда врач вынужден догадываться. Догадка оценивается по пятибалльной шкале (от 1 – определенно "нет", до 5 – определенно "да"). Врач вводит "4", так как он знает, что больной только что вернулся из экспедиции в пустыню Сахара. Система продолжает задавать вопросы. Когда все данные введены, ЭВМ ставит диагноз: гельминтозы – 80 %, малярия – 45 %.

Важно отметить, что современные экспертные системы основываются на знаниях экспертов. Системы не заставляют ЭВМ вырабатывать какие-либо правила самостоятельно. Такую функцию выполняют базы знаний, в которых процесс носит обратный характер. Программа работает с базой данных, где собраны различные факты, статистические данные об авариях на нефтяных вышках, описания принятых мер. Программа извлекает из информации, хранимой в базе данных, набор правил. Правила могут быть сформулированы в виде суждений на языке, близком к естественному, на основе которых можно принять решения, или могут быть записаны в виде программ, позволяющих ЭВМ вырабатывать решения автоматически.

Например, мы можем ожидать, что система базы знаний, в которой хранятся данные о торговых операциях фирмы, просмотрит записи о клиентах и попытается выявить характеристики потенциальных задолжников.

С помощью базы знаний, ориентированной на медицинские исследования, можно пытаться прогнозировать исход сердечных приступов или причины острых инфекционных заболеваний.

Во многих случаях базы знаний основываются на информации, соби-

раемой для решения обычных задач. Вывод правил на основе этой информации проводится параллельно обработке данных. Если правила окажутся полезными и надежными, то такое дополнительное использование вычислительной техники окажется весьма прибыльным, если нет – то это будет пустая трата времени.

В других случаях данные будут собираться в процессе научного исследования. Например, ученый исследует причины заболевания проказой в африканских деревнях. Он собирает данные об условиях жизни в деревнях, где встречаются случаи заболевания проказой: сколько людей пользуется одним колодцем, сколько в деревне собак, каковы санитарные условия, какая в данной местности растительность и т. д. Случается, что когда данные собраны, вывести соответствующее правило оказывается несложно. Между проказой и бродячими собаками существует связь. Однако часто правило (если оно существует) ускользает от исследователя. Программа искусственного интеллекта может быть единственным способом, позволяющим найти зависимость, скрытую в данных. Эта зависимость может оказаться совершенно неожиданной. Может оказаться, что проказа распространена в деревнях, где растет много растений, названия которых начинаются с буквы "б". Чем это объясняется?

На местном диалекте названия предметов красного цвета начинаются с буквы "б". Растения, которые растут в деревнях, пораженных проказой, произрастают на почве, богатой сурьмой, и имеют красный цвет. ЭВМ устанавливает, возможно, не известный до сих пор медикам факт: проказа связана с наличием в почве сурьмы.

В приведенном примере программное обеспечение базы знаний должно выполнять анализ символьных переменных с тем, чтобы найти объекты, начинающиеся с буквы "б". Существующие программы такую функцию реализуют пока не самым эффективным способом.

Как создать базу знаний на практике? В настоящее время мы не знаем, какой вариант базы знаний предпочтительнее и должны экспериментировать с рядом теоретических подходов, пока не убедимся, что один из них наилучшим образом использует реальную информацию и умение пользователей анализировать ее.

Эти подходы включают ряд альтернатив: от классического статистического анализа, бейесовского логического анализа, в котором используются не только ответы типа "да" и "нет", "истинно" или "ложно", но и теория вероятности, до генерации правила случайным образом.

Программа ИИ выполняет функцию, противоположную функции экспертной системы. Программа ищет повторяющиеся взаимосвязи в несвязанных, на первый взгляд, данных, в то время как экспертная система выполняет правила, сформулированные людьми.

Целесообразно начать с пакета программ под названием *Expertease*, разработанного специалистами кафедры ИИ Эдинбургского университета под руководством профессора Дональда Мичи. Пакет позволяет ввести в ЭВМ таблицы фактических данных и на основе бейесовской логики выводить из них правила, связывающие между собой две простые переменные.

Например, вы вводите в табличной форме результаты испытаний готовых деталей в механическом цехе. Масса, длина, ширина и высота каждой детали должны находиться в заданных пределах. Таблицы могут быть очень большими и насчитывать тысячи элементов. Программа отбирает в конечном итоге полдюжины основных проверок, включающих все другие проверки. В результате значительно снижается трудоемкость и стоимость технического контроля.

Однако до сих пор польза от такого пакета программ снижалась из-за отсутствия эффективной базы данных. Всю информацию приходилось вводить в ЭВМ специально. Если применить пакет *Expertease* в качестве интерфейса конечного пользователя к системе базы данных *Superfile*, то сбор данных, на основе которых выводятся новые зависимости, может быть значительно упрощен.

Пакет *Expertease* имеет дело с данными двух типов: "атрибутами", или характеристиками объектов информации, и "классами", или группировками, к которым относится объект, если к нему применить правило, сформулированное машиной.

В рассмотренном ранее примере к атрибутам относятся: число домов в деревне, средний размер семьи, проживающей в одном доме, расстояние до ближайшего водоема с питьевой водой, численность собак. (Система не просматривает названия растений, поэтому правило о начальной букве "б" не будет найдено). Все деревни, по которым имеются данные в базе, разбиваются программой по найденному правилу на две группы: с высокой и низкой вероятностью заболевания проказой.

Пользователь видит на экране форму, по которой в базу данных *Superfile* вводится информация и с ее помощью дает пакету *Expertease* задание просмотреть определенные поля и сформулировать правило, позволяющее предугадать значение других полей. Например, дежурный персонал в подразделениях интенсивной терапии вводит в систему следующие данные:

Больной[ ] Возраст [A]
Масса [ ] Рост [A]
Кровяное давление[ ] Пульс [A]
Выздоровление/летальный исход [K]

Система должна предсказать результат лечения больных с острым инфарктом миокарда. Программа просматривает поля, помеченные буквой "А" ("Атрибут"), и находит правило, в соответствии с которым заполня-

ется поле, помеченное буквой "К" ("Классы"). По окончании работы программы на экране может появиться правило, например следующее: (все цифры условные):

возраст
< 40: масса
> 18: летальный исход
< = 18: выздоровление
>= 40 кровяное давление
> 120: летальный исход
<= 120: пульс
> 100: летальный исход
< =100: выздоровление

Это значит, что для того чтобы ответить на вопрос, будет ли больной жить, сначала надо посмотреть на его возраст. Если возраст меньше 40 лет, то надо учитывать массу тела больного. При массе выше 18 стоунов (более 100 кг) больной обречен, при меньшей массе он выживет. Если возраст больного свыше 40 лет, имеет значение кровяное давление. При давлении более 120 человек не выживет. Если давление ниже, исход зависит от частоты пульса. При частоте пульса более 100 ударов в минуту человек умирает, при меньшей частоте пульса будет жить.

Программа *Expertease* работает довольно быстро. Она соотносит каждое значение атрибута с диапазоном значений этого атрибута, найденным при просмотре базы, и рассчитывает значение критерия, по которому ведется классификация объектов. Недостатком программы является то, что она не формирует правила, в основе которых лежат две или более переменных. Часто правила отражают взаимосвязи переменной со многими другими переменными внутри одной и той же записи, а не зависимость переменной от какой-то фиксированной величины.

Один из подходов был реализован в программе BEAGLE, разработанной Ричардом Форсайтом из Лондонского политехнического института [2]. Программа формулировала правила, набор операторов (больше, меньше, равно, плюс, минус, деления, умножения и т. д.), которые она применяла к полям записей, хранящихся в базовом файле. Она составляла правила и меняла их случайным образом. Каждое новое правило проверялось на реальных данных, и если оно работало лучше предыдущего — записывалось в память, а копия правила подвергалась новым изменениям случайным образом. Если правило работало хуже предыдущего, оно стиралось из памяти ЭВМ.

Р. Форсайт проверил BEAGLE на массиве записей по 100 больным, перенесшим сердечный приступ. По каждому пациенту, поступавшему в отделение интенсивной терапии, формировалось 16 показателей. В каждом случае ЭВМ выдавала прогноз. Программа выдавала 500 вариантов правил и остановилась на правиле, которое было справедливо в 81 % случаев. Правило формулируется следующим образом: если среднее артериальное давление (мм рт. ст.) больше или равно 61 минус суточное выделение мочи

(мл/ч), больной будет жить, иначе велика вероятность летального исхода. Ни один статистик не выведет такого рода правило, поскольку в нем сравниваются разнородные величины. Когда Форсайт сказал об этом правиле специалистам-медикам, первая реакция их была: "Чепуха!" а затем: "Может быть ...", поскольку хорошо известно, что перед смертью давление крови снижается, а выделение мочи увеличивается.

Однако даже такой подход не удовлетворяет всем практическим задачам. Представим себе базу данных фондовой биржи: каждая запись содержит динамику цен на акцию за определенный период времени. В этом случае программа, которая должна предсказать цену акций (информация, способная принести большую выгоду), вероятно, должна просмотреть цены всех акций на каждую дату, так как нас интересует разница между нашей акцией и общим индексом фондовой биржи. Безусловно, мы можем косвенным образом решить эту проблему, включив индекс в базу данных. Однако существуют более сложные задачи. Так, стоимость акций железнодорожных компаний может быть тесно взаимосвязана со стоимостью акций авиакомпаний.

Используя понятие "поле" и "запись", можно попытаться сгруппировать базы знаний в три категории:

1) базы, в которых осуществляется просмотр значений отдельных полей;

2) базы, в которых сравниваются значения полей внутри одной и той же записи;

3) базы, в которых сравниваются значения полей в разных записях.

7.1. ПРОБЛЕМЫ ИНФОРМАЦИИ

Человеческий мозг обладает непревзойденной способностью "выводить" правила из случайных данных. Возьмите, например, псевдонауки: астрологию и френологию. По определению, мы не знаем заранее, какие правила могут быть извлечены из базы данных, мы не знаем, существуют ли вообще какие-либо правила, а если существуют, то какого они рода. Любое правило, извлекаемое ЭВМ, может основываться только на данных, хранящихся в машине. Часто это вызывает трудности, которые человек легко мог бы разрешить. Например, ЭВМ может теряться в бесконечных поисках ответа на вопрос: почему в Африке не продаются меховые шубы? Возможно, ЭВМ потребуется длительное время, чтобы установить, что даты выписки накладных в Нью-Йорке приходятся на все дни недели, кроме субботы и воскресенья. Базы данных могут скрывать более сложные правила, чем те, которые доступны программе ИИ, хотя бы потому, что эти правила основываются также на информации, которой нет в базе. Например, (предполагается, что между проказой и сурьмой есть связь) маловероятно, что на присутствие сурьмы в почве укажет такая очевидная подсказка, как начальная буква "б" в названиях растений. Более вероятно, что машинные правила долгое время будут основываться лишь на малой доле того огромного числа случаев, котоыре составляют медицин-

щих в них элементах. Эта база должна находиться под управлением реляционной СУБД и ее обработка программой ИИ потребует огромных дополнительных затрат, так как надо проверять многочисленные гипотезы относительно цинка, серы, кадмия и т. д. В то же время следует отметить, что у нас нет никаких особых оснований обращаться дополнительно к базе данных о растениях. Даже если предположить, что базы данных, содержащие вспомогательную информацию, постоянно подключены к ЭВМ, ни один из существующих процессоров не способен просмотреть их достаточно быстро и извлечь истину, как это делает человеческий мозг в минуту озарения. Возможно в ближайшем будущем базы знаний будут объединяться с экспертными системами и человек сможет вмешаться в работу программы ИИ и исключить из рассмотрения субботы и воскресенья, когда речь идет о датах выписки транспортных документов, или наоборот, предложит дополнительно учесть данные об окраске растительности, когда проводится исследование заболевания в пустыне Сахара.

Объем данных, который можно использовать при нахождении правила, чрезвычайно велик. Поэтому нет абсолютно никакой уверенности, что компьютерная система, даже гипотетическая, может решить такую задачу. Вероятно, это объясняет, почему хорошие исследователи пользуются большим уважением. Непревзойденная способность человеческого мозга хранить и быстро извлекать огромные объемы информации становится главным препятствием для специалистов, работающих в области ИИ. "Интуицию" человека можно сравнить с поиском в базе данных по ключевым словам, которые по логике не имеют отношения к предмету поиска. Часто поиск этот весьма приближенный и связан с просмотром, выражаясь нашей терминологией, нескольких реляционных баз данных.

Способность мозга совершать мгновенные скачки через огромные массивы информации просто потрясающая. Известно, что лишь в редких случаях человек решает задачи методом перебора, просматривая большое число несвязных записей, пока не будет найдена нужная запись. Озарение либо наступает, либо не наступает, вот почему неудачная попытка установить с помощью базы знаний правило не означает, что такого правила вообще не существует. Другая проблема заключается в том, что люди могут безоговорочно верить тому, что выдает база знаний. Было бы наивным всегда следовать правилу, извлеченному из базы знаний. В противном случае у пользователя такой базы может выработаться определенный цинизм. База знаний, предугадавшая в 80 % случаев печальный исход для больных с острым инфарктом миокарда, через некоторое время может дать 100 % результат. В тех случаях, когда ЭВМ вынесет больному свой приговор, персонал может отнестись к нему, как человеку обреченному. Больной быстро перейдет в разряд неизлечимых, а правило получит новое подтверждение.

Хотя информация, выдаваемая ЭВМ в медицинских учреждениях, может выглядеть очень авторитетной и надежной, необходимо помнить, что машинные правила долгое время будут основываться лишь на малой доле того огромного числа случаев, которые составляют медицин-

скую практику, лежащую в основе обучения молодых врачей. В любых приложениях мы должны помнить, что ЭВМ лишь помощник нашему мозку и "интеллекта" в вычислительной машине не больше, чем в груде железок. У нее нет своих мыслей и она, к сожалению, подвержена ошибкам. И тем не менее, если воспринимать ЭВМ такой как она есть, можно получить очень полезного маленького помощника.

СПИСОК ЛИТЕРАТУРЫ

1. Date, C.J. (1981) An Introduction to Database Systems, Addison Wesley, Reading, Mass.
2. Forsyth, R. (1981) BEAGLE: A Darwinian approach to pattern recognition, Kybernetes, 10, 159–166.
3. Knuth, D. 619737 The Art of Computer Programming, vol. 3, Sorting and searching, Addison Wesley, Reading, Mass.
4. Laurie, P. 619837 The joy of Computers, Hutchinson and Little/Brown London/New York.

Производственное издание

ПИТЕР ЛОРИ

БАЗЫ ДАННЫХ ДЛЯ МИКРОЭВМ

Редактор *И.А. Сморчкова*
Художественный редактор *С.Н. Голубев*
Обложка художника *С.Н. Орлова*
Технический редактор *С.Ю. Синякова*
Корректор *Н.В. Давыдова*

ИБ № 5468

Сдано в набор 18.05.87.	Подписано в печать 08.01.88.	Формат 60×88 1/16.
Бумага офсетная № 2.	Гарнитура Пресс Роман.	Печать офсетная.
Усл. печ. л. 8,33.	Усл. кр.-отт. 8,58.	Уч.-изд. л. 9,33.
Тираж 30 000 экз.	Заказ. 1113	Цена 65 к.

Ордена Трудового Красного Знамени издательство "Машиностроение",
107076, Москва, Стромынский пер., 4

Отпечатано в Московской типографии № 4 Союзполиграфпрома
при Государственном комитете СССР по делам издательств, полиграфии
и книжной торговли 129041, Москва, Б. Переяславская, 46, с оригинала-макета,
изготовленного в издательстве "Машиностроение" на наборно-пишущих машинах

ПОПРАВКА

На стр. 133 последние 2 строки следует читать:

что подобную информацию можно извлечь только в том случае, если в систему будет интегрирована еще одна база данных – о растениях и входя-