УНИВЕРСИТЕТСКИЙ УЧЕБНИК

Серия «Прикладная математика и информатика»

С. Д. КУЗНЕЦОВ

БАЗЫ ДАННЫХ

Рекомендовано
учебно-методическим объединением
по классическому университетскому образованию
в качестве учебника для студентов
высших учебных заведений, обучающихся
по направлению подготовки «Прикладная
математика и информатика»

ACADEMIA

Москва
Издательский центр «Академия»
2012

УДК 621.38(075.8)
ББК 3281я733
К891

Рецензент —
канд. техн. наук, ст. науч. сотр., доц. *М.Р.Когаловский*
(зав. лабораторией систем баз данных Института проблем рынка РАН)

Кузнецов С. Д.

К891 Базы данных : учебник для студ. учреждений высшего проф. образования / С. Д. Кузнецов. — М. : Издательский центр «Академия», 2012. — 496 с. — (Университетский учебник. Сер. Прикладная математика и информатика).

ISBN 978-5-7695-8430-5

Учебник создан в соответствии с Федеральным государственным образовательным стандартом по направлению подготовки «Прикладная математика и информатика» (квалификация «бакалавр»).

В учебнике обсуждаются потребности разработчиков информационных систем в технологии баз данных, рассматриваются основные функции и типовая архитектура СУБД, а также приводится краткая характеристика нескольких популярных моделей данных. Подробно описывается реляционная модель данных, проектирование реляционных баз данных с использованием принципов нормализации и на основе семантических диаграммных моделей данных. В учебнике представлены также основные методы и алгоритмы, используемые в SQL-ориентированных СУБД; наиболее важные черты языка SQL как отдельной модели данных.

Для студентов учреждений высшего профессионального образования. Может быть использован студентами, обучающимися по направлениям подготовки «Информатика и вычислительная техника» и «Прикладная математика».

УДК 621.38(075.8)
ББК 3281я733

ISBN 978-5-7695-8430-5

ПРЕДИСЛОВИЕ

Хорошее знание и понимание технологии баз данных (БД) требуется любому специалисту в области программного обеспечения. Фундаментальное образование в этой области нельзя заменить тренингом. Как показывает практический опыт, люди, которые прошли тренинг-курсы, направленные на подготовку разработчиков приложений, проектировщиков, администраторов баз данных в среде конкретных систем управления базами данных (СУБД), но не получили предварительной базовой подготовки, обычно не оказываются способными достичь высоких профессиональных результатов. И наоборот, при наличии такой базовой подготовки люди зачастую быстро и качественно осваивают специфику конкретных систем без посторонней помощи. Поэтому нельзя переоценить важность обязательных курсов по технологии БД, читаемых в университетах и других высших учебных заведениях.

Данный учебник основан на материалах обязательного учебного курса, читаемого с 1994 г. автором на программистском потоке факультета вычислительной математики и кибернетики (ВМиК) Московского государственного университета им. М. В. Ломоносова.

За время своего существования курс неоднократно модифицировался при сохранении в целом его программы. Вначале основное внимание уделялось вопросам алгоритмов и методов построения СУБД, а модельно-языковые аспекты построения баз данных стояли на втором плане. Однако со временем стало ясно, что, во-первых, правильно понять и усвоить программистскую специфику организации СУБД можно только при наличии хорошего понимания модели данных, на которой основываются соответствующие базы данных, и языка, поддерживаемого СУБД для взаимодействия с этими базами данных. Во-вторых, очевидно, что среди выпускников ВМиК МГУ им. М. В. Ломоносова (как и других университетов и вузов) значительно больше специалистов, проектирующих базы данных и разрабатывающих их приложения, чем разработчиков СУБД.

Поэтому в последние годы модели данных и языковые средства заняли в курсе основное место, хотя по-прежнему значительная часть курса посвящается архитектуре СУБД и алгоритмам построения их наиболее важных подсистем. Данный учебник достаточно точно соответствует современному состоянию курса. Основная часть материала обычно укладывается в 68 ч (третий курс, первый семестр), а в спецкурсе (второй, весенний, семестр) рассматриваются более сложные аспекты моделей данных, описываемые, например, в [38]. От студентов, сдающих экзамен по спецкурсу, требуется полное знание материала основного курса.

Учебник состоит из 14 глав, сгруппированных в пять частей. В ч. I, вводной, содержится две главы. В гл. 1 обсуждается, каким образом потребности разработчиков информационных систем приводят к потребностям в системах управления базами данных, а также описываются типовая организация СУБД и ее основные подсистемы; гл. 2 содержит обзор семи моделей данных — трех ранних моделей, на которых основывались дореляционные СУБД (иерархическая и сетевая модели данных и модель инвертированных таблиц); классическая реляционная модель, введенная Эдгаром Коддом; и три современные модели данных (объектно-ориентированная модель данных, модель данных SQL и «истинно» реляционная модель, определенная Кристофером Дейтом и Хью Дарвеном).

Часть II также состоит из двух глав и посвящается развернутому описанию реляционной модели данных в авторской трактовке. В гл. 3 определяются основные понятия и термины реляционной модели, характеризуются три ее части и описываются структурная и целостная части; гл. 4 посвящена манипуляционной части реляционной модели данных. В ней обсуждаются два варианта реляционной алгебры — алгебра Кодда и современная, изящная, алгебра, введенная К. Дейтом и Х. Дарвеном, а также две разновидности реляционного исчисления — исчисление кортежей и доменное исчисление.

Часть III учебника, затрагивающую проблемы проектирования реляционных (и SQL-ориентированных) баз данных, составляют три главы. В гл. 5 и 6 описывается классический подход к проектированию реляционных баз данных на основе принципов нормализации. В гл. 5 излагаются основные понятия теории реляционных баз данных, связанные с функциональными зависимостями, и определяются вторая, третья нормальные формы и нормальная форма Бойса—Кодда. В гл. 6 вводится понятие многозначной зависимости, формулируются и дока-

зываются требуемые утверждения и определяется четвертая нормальная форма отношения. После этого определяются зависимость проекции/соединения и заключительная пятая нормальная форма, свойства которой невозможно улучшить на основе нормализации. В гл. 7 излагается материал по проектированию SQL-ориентированных баз данных на основе использования семантических диаграммных моделей — диаграмм «сущность-связь» и диаграмм классов языка UML.

В ч. IV книги, также состоящей из трех глав, описываются структуры данных, методы и алгоритмы, используемые в системах управления базами данных. В гл. 8 обсуждаются наиболее распространенные методы физической организации реляционных баз данных во внешней памяти, описываются применяемые для ускорения работы СУБД индексные структуры. В гл. 9 разобраны методы управления транзакциями — разновидностями двухфазного протокола синхронизационных блокировок; методы временны́х меток; методы, основанные на поддержке нескольких версий объектов базы данных. В гл. 10 описываются методы восстановления баз данных после различных сбоев на основе журнализации изменений базы данных.

Наконец, ч. V, последняя, состоящая из четырех глав, целиком посвящена языку SQL. Цель этой части состоит не в том, чтобы читатели могли выучить синтаксис языка и научиться писать простые запросы, а в том, чтобы показать, что язык SQL определяет полную и законченную модель данных, похожую на реляционную модель, но разительно от нее отличающуюся во многих важных аспектах. В гл. 11 обсуждаются система типов языка SQL, а также языковые средства определения базовых таблиц и ограничений целостности баз данных. В гл. 12 рассмотрен оператор выборки языка SQL; описаны синтаксис и семантика этого оператора, а также основные виды предикатов, которые можно использовать в условиях выборки. В гл. 13 изложены «поисковые» варианты операторов вставки, удаления и модификации таблиц в SQL-ориентированных базах данных, а также механизм триггеров. Наконец, в гл. 14, заключительной, описаны языковые средства управления транзакциями, сессиями и подключениями. В этой книге модель данных SQL представлена в сокращенной и неполной форме, позволяющей изложить соответствующий материал на лекциях. Более полное описание можно найти, например, в [38, 39].

Автор

ЧАСТЬ I
БАЗЫ ДАННЫХ, СУБД И МОДЕЛИ ДАННЫХ

Глава 1
НАЗНАЧЕНИЕ ТЕХНОЛОГИИ БАЗ ДАННЫХ. ФУНКЦИИ И ОСНОВНЫЕ КОМПОНЕНТЫ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

В данной главе определяется понятие информационной системы, обсуждаются предпосылки появления в компьютерах устройств внешней памяти, а также обосновывается принципиальная важность для организации информационных систем дисковых устройств с подвижными магнитными головками. Далее рассматриваются особенности организации и основное функциональное назначение одного из ключевых компонентов современных операционных систем — систем управления файлами. В третьем подразделе главы демонстрируется, что возможности файловых систем оказываются недостаточными для создания информационных программных систем и то, что естественные требования информационных систем к средствам управления данными во внешней памяти приводят к необходимости наличия систем управления базами данных (СУБД). В ходе этого анализа определяются основные функции, которые должны поддерживаться СУБД, и основные компоненты, которые содержит типичная СУБД.

1.1. Информационные системы и устройства внешней памяти

В общем смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного долговременного хранения информации в памяти компьютера, выполнении требуемых для данного приложения преобразований информации и/или вычислений, предостав-

лении пользователям системы удобного и легко осваиваемого интерфейса. Обычно объемы данных, с которыми приходится иметь дело таким системам, достаточно велики, а сами данные обладают достаточно сложной структурой. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

Надежное и долговременное хранение информации можно обеспечить только при наличии запоминающих устройств, сохраняющих информацию после выключения электропитания. Оперативная (основная) память этим свойством обычно не обладает. В первые десятилетия развития вычислительной техники использовались два вида устройств внешней памяти: магнитные ленты и магнитные барабаны. Емкость магнитных лент была достаточно велика, но по своей природе они обеспечивали только последовательный доступ к данным. Емкость магнитной ленты пропорциональна ее длине. Чтобы получить доступ к требуемой порции данных, нужно в среднем перемотать половину ее длины. Но чисто механическую операцию перемотки нельзя выполнить очень быстро. Поэтому быстрый произвольный доступ к данным на магнитной ленте, очевидно, невозможен.

Магнитный барабан представлял собой массивный металлический цилиндр с намагниченной внешней поверхностью и неподвижным пакетом магнитных головок. Такие устройства обеспечивали возможность достаточно быстрого произвольного доступа к данным, но позволяли сохранять сравнительно небольшой объем данных. Быстрый произвольный доступ осуществлялся благодаря высокой скорости вращения барабана и наличию отдельной головки на каждую дорожку магнитной поверхности; ограниченность объема была обусловлена наличием всего одной магнитной поверхности.

Указанные ограничения не очень существенны для систем численных расчетов. Обсудим более подробно, какие реальные потребности возникают у разработчиков таких систем. Прежде всего, для получения требуемых результатов серьезные вычислительные программы должны проработать достаточно долгое время (недели, месяцы и даже, может быть, годы). Наличие гарантий надежности со стороны производителей аппаратных компьютерных средств не избавляет программистов от необходимости использования программного сохранения частичных результатов вычислений, чтобы при возникновении непредвиденных сбоев аппаратуры можно было продолжить выполнение расчетов с некоторой контрольной точки. Для сохранения проме-

жуточных результатов идеально подходят магнитные ленты: при выполнении процедуры установки контрольной точки данные последовательно сбрасываются на ленту, а при необходимости перезапуска от сохраненной контрольной точки данные также последовательно с ленты считываются.

Вторая традиционная потребность программистов вычислительных приложений — максимально большой объем оперативной памяти. Большая оперативная память требуется, во-первых, для того, чтобы обеспечить программе быстрый доступ к большому количеству обрабатываемых данных. Во-вторых, сложные вычислительные программы сами могут иметь большой объем. Поскольку объем реально доступной в ЭВМ оперативной памяти всегда являлся недостаточным для удовлетворения текущих потребностей вычислений, требовалась быстрая внешняя память для организации оверлеев и/или виртуальной памяти. Здесь не будем вдаваться в детали организации этих механизмов программного расширения оперативной памяти, но заметим, что для этого идеально подходили магнитные барабаны. Они обеспечивали быстрый доступ к внешней памяти, а для расширения оперативной памяти одной программы (сложные вычислительные программы, как правило, выполняются на компьютере в одиночку) большой объем внешней памяти не требуется.

Далее заметим, что, даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти, чтобы программа работала как можно быстрее. Развитая поддержка работы с внешней памятью со стороны общесистемных программных средств не обязательна, а иногда и вредна, поскольку приводит к дополнительным накладным расходам аппаратных ресурсов.

Однако для информационных систем, в которых объем постоянно хранимых данных определяется спецификой бизнес-приложения, а потребность в текущих данных — пользователем приложения, одних только магнитных барабанов и лент недостаточно. Емкость магнитного барабана просто не позволяет долговременно хранить данные большого объема. Что же касается лент, то представьте себе состояние человека у билетной кассы, который должен дождаться полной перемотки магнитной ленты. Естественным требованием к таким системам является обеспечение высокой средней скорости выполнения операций при наличии больших объемов данных.

Именно требования к устройствам внешней памяти со стороны бизнес-приложений вызвали появление устройств внешней

памяти со съемными пакетами магнитных дисков и подвижными головками чтения/записи, что явилось революцией в истории вычислительной техники. Эти устройства памяти обладали существенно большей емкостью, чем магнитные барабаны (за счет наличия нескольких магнитных поверхностей), обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь архив данных практически неограниченного объема.

Магнитные диски представляют собой пакеты магнитных пластин (поверхностей), между которыми на одном рычаге движется пакет магнитных головок (рис. 1.1). Шаг движения пакета головок является дискретным, и каждому положению пакета головок логически соответствует цилиндр пакета магнитных дисков. На каждой поверхности цилиндр «высекает» дорожку, так что каждая поверхность содержит число дорожек, равное числу цилиндров. При разметке магнитного диска (специальном действии, предшествующем использованию диска) каждая дорожка размечается на одно и то же количество блоков; таким образом, предельная емкость каждого блока составляет одно и то же число байтов. Для произведения обмена с магнитным диском на уровне аппаратуры нужно указать номер цилиндра, номер поверхности, номер блока на соответствующей дорожке и число байтов, которое нужно записать или прочитать от начала этого блока.

При выполнении обмена с диском аппаратура выполняет три основных действия:

- подвод головок к нужному цилиндру (обозначим время выполнения этого действия как $t_{\text{пг}}$);
- поиск на дорожке нужного блока (время выполнения — $t_{\text{пб}}$);
- собственно обмен с этим блоком (время выполнения — $t_{\text{об}}$).

Тогда, как правило, $t_{\text{пг}} \gg t_{\text{пб}} \gg t_{\text{об}}$, потому что подвод головок — это механическое действие, причем в среднем нужно переместить головки на расстояние, равное половине радиуса

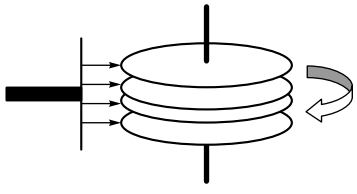

Рис. 1.1. Грубая схема дискового устройства памяти с подвижными головками

поверхности, а скорость передвижения головок не может быть слишком большой по физическим соображениям. Поиск блока на дорожке требует прокручивания пакета магнитных дисков в среднем на половину длины внешней окружности; скорость вращения диска может быть существенно больше скорости движения головок, но она тоже ограничена законами физики. Для выполнения же обмена нужно прокрутить пакет дисков всего лишь на угловое расстояние, соответствующее размеру блока. Таким образом, из всех этих действий в среднем наибольшее время занимает первое, и поэтому существенный выигрыш в суммарном времени обмена при считывании или записи только части блока получить практически невозможно.

С появлением магнитных дисков началась история систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволял или очень затруднял поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

1.2. Файловые системы

Историческим шагом явилось появление систем управления файлами. С точки зрения программиста приложений — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса внешней памяти и обеспечение доступа к данным.

В этом подразделе будут обсуждаться история файловых систем, их основные черты и области разумного применения. Однако сначала уместно сделать два замечания. Во-первых, заметим, что в области управления файлами исторически

существует некоторая терминологическая путаница. Термин *файловая система* (*file system*) используется для обозначения как программной системы, управляющей файлами, так и архива файлов, хранящегося во внешней памяти. Было бы лучше в первом случае использовать термин *система управления файлами*, оставив за термином *файловая система* только второе значение. Однако принятая практика вынуждает использовать в этой книге термин *файловая система* в обоих смыслах. Точный смысл термина должен быть понятен из контекста. (Аналогичная путаница возникает при некорректном использовании терминов *база данных* и *система управления базами данных*. В данной книге эти термины строго различаются.) Во-вторых, для целей этой главы достаточно ограничиться описанием основных свойств так называемых традиционных файловых систем, не затрагивая особенности современных систем с повышенной надежностью.

Первая развитая файловая система была разработана специалистами IBM в середине 1960-х гг. для выпускавшейся компанией серии компьютеров System/360. В этой системе поддерживались как чисто последовательные, так и индексно-последовательные файлы (а также файлы с прямым доступом к записям), а реализация во многом опиралась на возможности только появившихся к этому времени контроллеров управления дисковыми устройствами. Контроллеры обеспечивали возможность обмена с дисковыми устройствами порциями данных произвольного размера, а также индексный доступ к записям файлов, и эти функции контроллеров активно использовались в файловой системе OS/360.

Файловая система OS/360 обеспечила будущих разработчиков уникальным опытом использования дисковых устройств с подвижными головками, который отражается во всех современных файловых системах.

1.2.1. Структуры файлов

Практически во всех современных компьютерах основными устройствами внешней памяти являются магнитные диски с подвижными головками, и именно они служат для хранения файлов. Как отмечалось ранее, аппаратура магнитных дисков допускает выполнение обмена с дисками порциями данных произвольного размера. Однако возможность обмениваться с магнитными дисками порциями, размеры которых меньше полного объема блока, в настоящее время в файловых системах не используется. Это связано с двумя обстоятельствами.

Во-первых, считывание или запись только части блока не приводит к существенному выигрышу в суммарном времени обмена, поскольку, как указывалось в подразд. 1.1, основная часть этого времени тратится на перемещение магнитных головок. Во-вторых, для работы с частями блоков файловая система должна обеспечить буферы оперативной памяти соответствующего размера, что существенно усложняет распределение основной памяти. Алгоритмы распределения памяти порциями произвольного размера плохи тем, что любой из них рано или поздно приводит к *внешней фрагментации* памяти. В памяти образуется большое число мелких свободных фрагментов. Их совокупный размер может быть больше размера любого требуемого буфера, но его можно выделить, только если произвести сжатие памяти, т. е. подвижку всех занятых фрагментов таким образом, чтобы они располагались вплотную один к другому. Во время выполнения операции сжатия памяти нужно приостановить выполнение обменов, а сама эта операция занимает много времени.

Поэтому во всех современных файловых системах явно или неявно выделяется уровень, обеспечивающий работу с *базовыми файлами*, которые представляют собой наборы блоков, последовательно нумеруемых в адресном пространстве файла и отображаемых на физические блоки диска (рис. 1.2). Размер логического блока файла совпадает с размером физического блока диска или кратен ему; обычно размер логического блока выбирается равным размеру страницы виртуальной памяти, поддерживаемой аппаратурой компьютера совместно с операционной системой.

В некоторых файловых системах базовый уровень был доступен пользователю, но чаще он прикрывался некоторым более

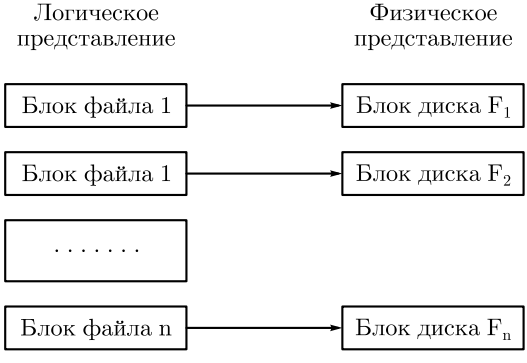

Рис. 1.2. Схематичное изображение базового файла

высоким уровнем, стандартным для пользователей. Исторически существует два основных подхода. При первом подходе, свойственном, например, файловой системе операционной системы компании Hewlett-Packard OpenVMS, пользователи представляют файл как последовательность записей. Каждая запись — это последовательность байтов, имеющая постоянный или переменный размер. Можно читать или писать записи последовательно либо позиционировать файл на запись с указанным номером.

В некоторых файловых системах допускается структуризация записей на поля и объявление указываемых полей ключами записи. В таких файловых системах можно потребовать выборку записи из файла по ее заданному ключу. Естественно, в этом случае файловая система поддерживает в том же (или другом, служебном) базовом файле дополнительные, невидимые пользователю, служебные структуры данных — *индексы*. Распространенные способы организации индексов ключевых файлов основаны на технике *хэширования* и *B-деревьев* (подробно эта техника обсуждается в гл. 8 в контексте физической организации хранения баз данных). Существуют и многоключевые способы организации файлов (у одного файла объявляется несколько ключей, и можно выбирать записи по значению каждого ключа).

Второй подход, получивший распространение вместе с операционной системой UNIX, состоит в том, что любой файл представляется как непрерывная последовательность байтов. Из файла можно прочитать указанное число байтов, либо начиная с его начала, либо предварительно выполнив его позиционирование на байт с указанным номером. Аналогично можно записать указанное число байтов либо в конец файла, либо предварительно выполнив позиционирование файла. Тем не менее заметим, что скрытым от пользователя, но существующим во всех разновидностях файловых систем ОС UNIX является базовое блочное представление файла.

Конечно, в обоих случаях можно обеспечить набор преобразующих функций, приводящих представление файла к другому виду. Примером тому может служить поддержка стандартной файловой среды UNIX в среде операционной системы OpenVMS.

1.2.2. Логическая структура файловых систем и именование файлов

Во всех современных файловых системах обеспечивается многоуровневое именование файлов за счет наличия во внешней памяти каталогов — дополнительных файлов со специальной

структурой. Каждый каталог содержит имена каталогов и/или файлов, хранящихся в данном каталоге. Таким образом, полное имя файла состоит из списка имен каталогов плюс имя файла в каталоге, непосредственно содержащем данный файл.

Поддержка многоуровневой схемы именования файлов обеспечивает несколько преимуществ, основным из которых является простая и удобная схема логической классификации файлов и генерации их имен. Можно сопоставить каталог или цепочку каталогов с пользователем, подразделением, проектом и затем образовывать в этом каталоге файлы или каталоги, не опасаясь коллизий с именами других файлов или каталогов.

Разница между способами именования файлов в разных файловых системах состоит в том, с чего начинается эта цепочка имен. В любом случае первое имя должно соответствовать корневому каталогу файловой системы. Вопрос заключается в том, как сопоставить этому имени корневой каталог — где его искать? В связи с этим имеются два радикально различных подхода.

Во многих системах управления файлами требуется, чтобы каждый архив файлов (полное дерево каталогов) целиком располагался на одном дисковом пакете или логическом диске — разделе физического дискового пакета, логически представляемом в виде отдельного диска с помощью средств операционной системы. В этом случае полное имя файла начинается с имени дискового устройства, на котором установлен соответствующий диск. Такой способ именования использовался в файловых системах компаний IBM и DEC; очень близки к этому и файловые системы, реализованные в операционных системах семейства Windows компании Microsoft. Можно назвать такую организацию поддержкой изолированных файловых систем.

Другой крайний вариант был реализован в файловых системах операционной системы Multics. Эта система заслуживает отдельного разговора, в ней был реализован целый ряд оригинальных идей, но мы остановимся только на особенностях организации архива файлов. В файловой системе Multics пользователям обеспечивалась возможность представлять всю совокупность каталогов и файлов в виде единого дерева. Полное имя файла начиналось с имени корневого каталога, и пользователь не обязан был заботиться об установке на дисковое устройство каких-либо конкретных дисков. Сама система, выполняя поиск файла по его имени, запрашивала у оператора установку необходимых дисков. Такую файловую систему можно назвать полностью централизованной.

Конечно, во многом централизованные файловые системы удобнее изолированных: система управления файлами выполняет больше рутинной работы. В частности, администратор файловой системы автоматически оповещается о потребности установки требуемых дисковых пакетов; система обеспечивает равномерное распределение памяти на известных ей дисковых томах; возможна организация автоматического перемещения редко используемых файлов на более медленные носители внешней памяти; облегчается рутинная работа, связанная с резервным копированием.

Но в таких системах возникают существенные проблемы, если требуется перенести поддерево файловой системы на другую вычислительную установку. Поскольку файлы и каталоги любого логического поддерева могут быть физически разбросаны по разным дисковым пакетам и даже магнитным лентам, для такого переноса требуется специальная утилита, собирающая все объекты требуемого поддерева на одном внешнем носителе, не входящем в состав штатных устройств централизованной файловой системы. Конечно, даже при наличии такой утилиты выполнение процедуры физической сборки требует существенного времени.

Компромиссное решение применяется в файловых системах ОС UNIX. На базовом уровне в этих файловых системах поддерживаются изолированные архивы файлов. Один из таких архивов объявляется корневой файловой системой. Это делается на этапе генерации операционной системы, и после запуска операционная система «знает», на каком дисковом устройстве (физическом или логическом) располагается корневая файловая система. После запуска системы можно «смонтировать» корневую файловую систему и ряд изолированных файловых систем в одну общую файловую систему. Технически это осуществляется посредством создания в корневой файловой системе специальных пустых каталогов (точек монтирования).

Специальный системный вызов *mount* ОС UNIX позволяет подключить к одному из пустых каталогов корневой каталог указанного архива файлов. Выполнение такого действия приводит к «наложению» корневого каталога монтируемой файловой системы на каталог точки монтирования; корневой каталог приобретает имя каталога точки монтирования. После монтирования общей файловой системы именование файлов производится так же, как если бы она с самого начала была централизованной. Если учесть, что обычно монтирование файловой системы производится при раскрутке системы (при

выполнении стартового командного файла), пользователи ОС UNIX, как правило, и не задумываются о происхождении общей файловой системы.

Кроме того, поддерживается системный вызов unmount, «отторгающий» ранее смонтированную файловую систему от общей иерархии. Конечно, все это заметно облегчает перенос частей файловой системы на другие установки.

1.2.3. Авторизация доступа к файлам

Поскольку файловая система является общим хранилищем файлов, принадлежащих, вообще говоря, разным пользователям, системы управления файлами должны обеспечивать *авторизацию* доступа к файлам. В общем виде подход состоит в том, что для каждого зарегистрированного пользователя и для каждого существующего файла файловой системы указываются действия, выполнение которых над данным файлом разрешено или запрещено данному пользователю (так называемый мандатный способ защиты — у каждого пользователя имеется или не имеется отдельный мандат для работы с каждым файлом). Применение мандатного способа авторизации доступа влечет за собой существенные накладные расходы, связанные с потребностью хранения избыточной информации и использованием этой информации для проверки правомочности доступа.

Поэтому в большинстве современных систем управления файлами применяется упрощенный подход к *авторизации* доступа к файлам, традиционно поддерживаемый, например в ОС UNIX (так называемый *дискреционный* подход). При использовании этого подхода с каждым зарегистрированным пользователем связывается пара целочисленных идентификаторов: идентификатор группы, к которой относится пользователь, и его собственный идентификатор. Этими же идентификаторами снабжается каждый процесс, запущенный от имени данного пользователя и имеющий возможность обращаться к системным вызовам файловой системы. Соответственно, при каждом файле хранится полный идентификатор пользователя (собственный идентификатор плюс идентификатор группы), который создал этот файл, и помечается, какие действия с файлом может производить он сам, какие действия с файлом доступны для остальных пользователей той же группы и что могут делать с файлом пользователи других групп. Для каждого файла контролируется возможность выполнения трех действий: чтение, запись и выполнение. Хранимая информация очень компактна (два целых

числа для представления идентификаторов и шкала из 9 бит для характеристики возможных действий), при проверке требуется небольшое количество действий, и этот способ контроля доступа в большинстве случаев удовлетворителен.

1.2.4. Синхронизация многопользовательского доступа

Если операционная система поддерживает многопользовательский режим, может возникнуть ситуация, когда два или более пользователей (процессов) одновременно пытаются работать с одним и тем же файлом. Если все эти пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этой группы требуется взаимная синхронизация.

В файловых системах обычно применяется следующий подход. В операции открытия файла (первой и обязательной операции, с которой должен начинаться сеанс работы с файлом) помимо прочих параметров указывается режим работы (чтение или изменение). Если к моменту выполнения этой операции от имени некоторого процесса A файл уже открыт некоторым другим процессом B, причем существующий режим открытия несовместим с требуемым режимом (совместимы только режимы чтения), то в зависимости от особенностей системы либо процессу A сообщается о невозможности открытия файла в нужном режиме, либо процесс A блокируется до тех пор, пока процесс B не выполнит операцию закрытия файла.

1.2.5. Области разумного применения файлов

После краткого обзора технологии файловых систем обсудим возможные области их применения. Чаще всего файлы используются для хранения текстовых данных: документов, текстов программ и т. д. Такие файлы обычно создаются и модифицируются с помощью различных текстовых редакторов. Эти редакторы могут быть очень простыми, такими, как *ed* в мире UNIX или утилиты редактирования (*Far Manager*, *WordPad* и т. д.) в среде Windows. Они могут быть сложными и многофункциональными, синтаксически ориентированными, как, например, *GNU Emacs*. Но обычно структура текстовых файлов очень проста (с точки зрения файловой системы): это либо последовательность записей, содержащих строки текста, либо последовательность байтов, среди которых встречаются специальные символы (например, символы конца строки). Конечно же, сложность логической

структуры текстового файла определяется текстовым редактором, но в любом случае файловой системе она не видна.

Файлы, содержащие тексты программ, используются как входные файлы компиляторов (чтобы правильно воспринять текст программы, компилятор должен понимать логическую структуру текстового файла), которые, в свою очередь, формируют файлы, содержащие объектные модули. С точки зрения файловой системы объектные файлы также обладают очень простой структурой — последовательность записей или байтов. Система программирования накладывает на такую структуру более сложную и специфичную для этой системы логическую структуру объектного модуля. Подчеркнем, что логическая структура объектного модуля файловой системе неизвестна; эта структура поддерживается инструментами системы программирования.

Аналогично обстоит дело с файлами, формируемыми редакторами связей (редактор связей должен понимать логическую структуру файлов объектных модулей) и содержащими образы выполняемых программ. Логическая структура таких файлов остается известной только редактору связей и загрузчику — программе операционной системы. Общая схема взаимодействия программных компонентов при построении программы показана на рис. 1.3. Мы кратко обозначили способы использования файлов в процессе разработки программ, но можно сказать, что ситуация аналогична и в других случаях: например, при создании и использовании файлов, содержащих графическую, аудио- и видеоинформацию.

Одним словом, файловые системы обычно обеспечивают хранение данных с очень «аморфной» стандартной структурой, оставляя дальнейшую структуризацию прикладным программам. В перечисленных ранее случаях использования файлов это

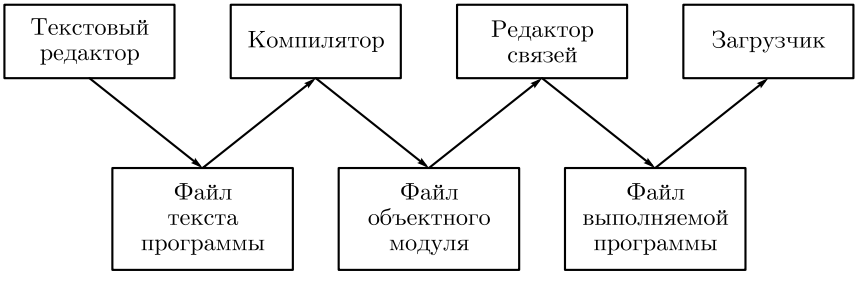

Рис. 1.3. Связи между программными компонентами по пониманию логической структуры файлов

даже хорошо, поскольку при разработке любой новой прикладной системы, опираясь на простые, стандартные и сравнительно дешевые средства файловой системы, можно реализовать те структуры хранения, которые наиболее точно соответствуют специфике данной прикладной области.

1.3. Потребности информационных систем

Удовлетворяют ли рассмотренные ранее базовые возможности файловых систем потребности информационных систем? Типовая информационная система, главным образом, ориентирована на хранение, выбор и модификацию данных соответствующей прикладной области. Структура таких данных зачастую очень сложна, и, хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего.

На начальном этапе использования вычислительной техники для построения информационных систем проблемы структуризации данных решались индивидуально в каждой информационной системе. Производились необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т. д. (рис. 1.4).

Но поскольку для функционирования информационных систем требуются сложные структуры данных, эти дополнительные индивидуальные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Стремление выделить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось, на мой взгляд, первой побудительной причиной создания СУБД. Очень скоро стало понятно, что невозможно обойтись общей библиотекой программ (рис. 1.5), реализующей над стандартной базовой файловой системой более сложные методы хранения данных.

Рис. 1.4. Примитивная схема структуризации данных в информационной системе