ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Э. В. ФУФАЕВ, Д. Э. ФУФАЕВ

БАЗЫ ДАННЫХ

Рекомендовано
Федеральным государственным автономным учреждением
«Федеральный институт развития образования»
в качестве учебного пособия для использования в учебном процессе
образовательных учреждений, реализующих программы
среднего профессионального образования по специальности
«Информационные системы (по отраслям)»

Регистрационный номер рецензии 504
от 11 декабря 2014 г. ФГАУ «ФИРО»

10-е издание, стереотипное

ACADEMA

Москва
Издательский центр «Академия»
2015

УДК 621.38(075.32)
ББК 3281я723
Ф94

Рецензенты:
зам. декана факультета «Информатика и телекоммуникации» Московского государственного института электроники и математики, канд. техн. наук, доцент кафедры «Лазерные микроволновые информационные системы» *Д. П. Николаев*;
преподаватель Московского государственного колледжа информационных технологий *И. А. Кумскова*

Фуфаев Э. В.

Ф94 Базы данных : учеб. пособие для студ. учреждений сред. проф. образования / Э. В. Фуфаев, Д. Э. Фуфаев. — 10-е изд., стер. — М. : Издательский центр «Академия», 2015. — 320 с.

ISBN 978-5-4468-2436-6

Учебное пособие создано в соответствии с требованиями Федерального государственного образовательного стандарта среднего профессионального образования по специальности «Информационные системы (по отраслям)»; ОП «Основы проектирования баз данных».

Изложены теоретические основы проектирования баз данных и методология их практического применения в процессах принятия решений при управлении производством и бизнесом.

Для студентов учреждений среднего профессионального образования.

УДК 621.38(075.32)
ББК 3281я723

Учебное издание

Фуфаев Эдуард Валентинович, Фуфаев Дмитрий Эдуардович

Базы данных

Учебное пособие

10-е издание, стереотипное
Редактор *И. В. Мочалова*. Технический редактор *Е. Ф. Коржуева*
Компьютерная верстка: *М. В. Трушина*. Корректор *Т. В. Кузьмина*

Изд. № 1101000111. Подписано в печать 07.07.2015. Формат 60×90/16. Гарнитура «Таймс». Печать офсетная. Бумага офсетная № 1. Усл. печ. л. 20,0. Тираж 1 000 экз. Заказ № 37247.

ООО «Издательский центр «Академия», www.academia-moscow.ru
129085, Москва, пр-т Мира, 101В, стр. 1.
Тел./факс: (495) 648-0507, 616-00-29.
Санитарно-эпидемиологическое заключение № РОСС RU. АЕ51. Н 16679 от 25.05.2015.

Отпечатано в соответствии с качеством предоставленных издательством электронных носителей в ОАО «Саратовский полиграфкомбинат».
410004, г. Саратов, ул. Чернышевского, 59. www.sarpk.ru

ISBN 978-5-4468-2436-6

ОТ АВТОРОВ

Целью экономической деятельности любой фирмы как в сфере материального производства, так и в сфере обслуживания населения является получение прибыли.

Достижение этой цели — задача многоплановая, зависящая от творческих способностей всех сотрудников фирмы. Решить поставленную задачу можно лишь в том случае, если каждый участник производственного процесса — от руководителя до рядового исполнителя — сможет принимать оптимальные решения на своем участке производства. Принимать оптимальные решения с первого раза, а не исправлять последствия неправильных решений — вот главная задача любого специалиста. Эффективно решить эту задачу можно с применением компьютерных информационных систем, разрабатываемых на основе баз данных.

Базы данных как одно из направлений теории информации представляют собой методы и средства разработки компьютерных информационных систем, основу которых составляют особым образом структурированные файлы, предоставляющие пользователю эффективные методы получения и анализа данных, необходимых для принятия оптимального решения.

Теория разработки баз данных — сравнительно молодая область науки, однако базы данных являются сегодня основой таких направлений в разработке автоматизированных систем обработки информации, как системы искусственного интеллекта, экспертные системы, системы автоматизированного конструкторского и технологического проектирования.

Учебник состоит из пяти частей.

Часть I посвящена теоретическим основам проектирования и организации баз данных.

В части II изложена технология разработки баз данных с применением прикладной программной системы Access фирмы Microsoft, языков программирования Visual Basic, SQL.

В части III рассмотрены теоретические основы разработки систем управления распределенными базами данных и дан обзор программных систем SQL Server 2000 и Oracle.

В части IV рассмотрены основные тенденции развития теории проектирования баз данных, так называемые постреляционные базы данных.

В части V даны конкретные примеры применения баз данных в задачах управления современным производством и бизнесом.

Главы 1—5, 7—9 написаны Фуфаевым Дмитрием Эдуардовичем, инженером-программистом 1-й категории; гл. 6, 10—18 — канд. техн. наук, доцентом Фуфаевым Эдуардом Валентиновичем.

Выражаем благодарность сотрудникам Отдела главного технолога ОАО «Раменский приборостроительный завод».

ВВЕДЕНИЕ

Конец XX — начало XXI в. характеризуются активным внедрением в деятельность человечества компьютерных информационных технологий, особенно систем управления базами данных (СУБД). СУБД — это программные системы управления структурированными файлами данных, обеспечивающих пользователю оперативное получение необходимой информации.

Структурированные файлы данных, или базы данных, являются неотъемлемой частью автоматизированных систем управления (АСУ), систем искусственного интеллекта и экспертных систем, систем автоматизированного проектирования конструкторской документации САПР-КД или CAD-систем (Computer Aided Design), систем автоматизированного проектирования технологических процессов изготовления изделий САПР-ТП или CAM-систем (Computer Aided Manufacturing).

Прообразом современных СУБД можно считать первые отечественные теоретические и практические работы по решению на ЭВМ информационно-логических задач, т.е. задач хранения, обработки и получения информации по определенным наборам логических правил.

В этих работах, проводимых в начале 60-х гг. XX в., решались задачи оптимального распределения ресурсов ЭВМ, быстроты поиска, обеспечения достоверности информации, создания языков программирования. Данные работы были ориентированы на серийно выпускаемые отечественные электронно-вычислительные машины типа «Минск-2». В то время эта машина относилась к классу машин средней производительности и имела следующие характеристики:

Быстродействие	5—6 тыс. операций в секунду
Объем ОЗУ	18 Кбайт
Объем памяти внешнего накопителя на магнитном диске	2 Мбайт

Машина «Минск-2» занимала площадь 40...50 м^2.

Алгоритмы обработки информационно-логических задач основывались на элементах алгебры логики, которые сегодня называют реляционной алгеброй. В качестве языков программирования

применялись языки АЛГОЛ-60 и разработанный отечественными специалистами язык АЛГЭМ.

На этих программных и аппаратных средствах создавались следующие информационные системы:

- обработка массивов записей, основу которой составляли процессы сортировки и перегруппировки данных. Записью называли точно установленный набор данных, характеризующих некоторый объект или процесс;
- накопление и поиск данных в иерархических классификационных системах, содержащих операции пополнения и удаления данных, поиска сведений по запросам;
- накопление и анализ фактографических данных, основным принципом построения которых являлось смысловое кодирование информации.

Такие информационные системы и соответствующие языки программирования включали в себя четыре основные части, которые составляют основу и современных баз данных:

- набор терминов, обозначающих основные предметы конкретной области знаний;
- набор смысловых отношений между предметами;
- набор формальных правил (синтаксис), позволяющий из основных терминов и отношений языка строить более сложные термины и отношения;
- систему кодирования понятий, отношений и синтаксических правил языка, позволяющую осуществлять преобразование и хранение информации.

Среди зарубежных исследований на данном периоде развития информационных технологий следует выделить американскую компанию IBM, в которой в 1968 г. была введена в эксплуатацию промышленная СУБД на ЭВМ типа IBM 360.

Настоящий прорыв в области теории и практики создания информационных систем на основе баз данных наступил вместе с эрой персональных компьютеров и созданием теории реляционных баз данных.

Главной особенностью этого периода является появление персональных компьютеров с дисковой операционной системой DOS и разработка специальных языков для проектирования баз данных с элементами унификации: семейство языков Dbase, FoxPro, Clipper.

Дальнейшее развитие СУБД связано со стандартизацией в области программного обеспечения на основе структурированного языка запросов SQL (Structured Query Language).

В настоящее время базы данных составляют основу нового направления в совершенствовании как бизнес-процессов, так и процессов проектирования, изготовления и эксплуатации сложных наукоемких изделий — создание единого информационного про-

странства, сопровождающего весь жизненный цикл продукции. Это направление называют CALS-технологиями.

Сущность CALS-технологий сводится к созданию единого информационного пространства, обеспечивающего возможность получения достоверной информации, необходимой для принятия оптимальных решений на всем жизненном цикле изделия, начиная от его проектирования и заканчивая утилизацией после окончания сроков эксплуатации.

Процесс принятия оптимального решения в любых сферах деятельности связан прежде всего с необходимостью предварительного анализа больших объемов информации, с тем чтобы сначала найти область допустимых решений, а затем в ограниченной области найти одно единственное, оптимальное, решение.

Как показала практика, принятие нерациональных решений во многом связано с отсутствием у специалиста необходимой информации.

Базы данных и системы управления базами данных в этом случае можно рассматривать как стратегические средства в совершенствовании управления производством.

«Принимать оптимальные решения с первого раза, а не устранять результаты и последствия неправильных технических решений» — вот девиз управленческой деятельности современных предприятий. СУБД — это успех в управлении.

ЧАСТЬ I

ТЕОРИЯ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Глава 1

АВТОМАТИЗИРОВАННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ НА ОСНОВЕ БАЗ ДАННЫХ

1.1. Базы данных, системы управления базами данных

Развитие современного промышленного производства и бизнеса невозможно без создания автоматизированных информационных систем, одно из назначений которых — предоставление пользователю достоверной информации, необходимой для принятия оптимального решения. В настоящее время ни одна из задач управления производством и бизнесом не должна выполняться без применения автоматизированных информационных систем. Это анализ рынка и проектирование конструкции и технологии изготовления новых изделий; это системы управления производственными, технологическими процессами и качеством изготовления продукции.

Сегодня мы должны рассматривать любую деятельность любого специалиста как некоторую систему принятия решений, поэтому специалисту и нужна достоверная информация. Таким образом, одной из важнейших функций информационной системы является информационное обеспечение процесса управления. Такие системы называют управленческими информационными системами (management information systems); они обычно включают в себя большие и сложные базы данных.

Итак, что же такое *База данных* и *Система управления базами данных*? К сожалению, в большинстве книг по этому направлению информационных технологий нет достаточно четких определений. Рассмотрим некоторые из них.

В учебном пособии Карповой Т.С. [6] даны следующие определения.

База данных (*БД*) — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

Система управления базами данных — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

В руководстве по настольной СУБД Mcrosoft Access 2000 авторы Харитонова И.А. и Михеева В.Д. [13] предлагают следующие определения.

База данных — это совокупность сведений (о реальных объектах, процессах, событиях или явлениях), относящихся к определенной теме или задаче, организованная таким образом, чтобы обеспечить удобное представление этой совокупности как в целом, так и любой ее части.

Основные функции Системы управления базами данных — это определение данных (описание структуры базы данных), обработка данных и управление данными.

Обратим внимание на существенное отличие второго определения БД, а именно на характеристику особой организации информации в БД.

В наибольшей степени, по нашему мнению, понятия базы данных и СУБД определены в Толковом словаре по вычислительным системам [11].

База данных, в обычном, строгом смысле слова — файл данных, для определения и обращения к которому используются средства управления базой данных. Это означает, во-первых, что этот файл определен посредством схемы, не зависящей от программ, которые к нему обращаются, и, во-вторых, что он реализован в виде запоминающего устройства с прямым доступом.

С учетом данных определений можно сказать, что *База данных* — это файл, организованный (структурированный) как файл с прямым доступом.

Понятие файла прямого доступа и принципов его организации известны из теории программирования многомерных массивов, в том числе на алгоритмических языках Basic, Паскаль.

В Толковом словаре по вычислительным системам [11] даны следующие понятия СУБД.

Система управления базой данных — система программного обеспечения, имеющая средства обработки на языке базы данных, позволяющая обрабатывать обращения к базе данных, которые поступают от прикладных программ и (или) конечных пользователей, и поддерживать целостность базы данных.

Обычно различают три класса (модели) организации баз данных: иерархические, сетевые и реляционные. Термин «модель» в данном случае рассматривается как структура, позволяющая количественно и качественно оценивать на логическом уровне организацию хранения и доступа к данным (например, рассчитать ожидаемую потребность в памяти для хранения данных или рассчитать потребное число шагов поиска данных).

Иерархическая модель данных. Иерархическая модель данных, как следует из названия, имеет иерархическую структуру, т.е. каждый из элементов связан только с одним стоящим выше элемен-

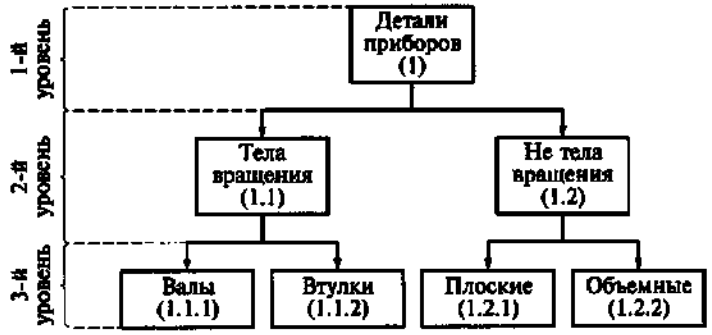


Рис. 1.1. Иерархическая модель данных

том, но в то же время на него могут ссылаться один или несколько стоящих ниже элементов. В терминологии иерархической модели используются более конкретные понятия: «элемент» (узел); «уровень» и «связь». *Узел* чаще всего представляет собой атрибут (признак), описывающий некоторый объект. Иерархически модель схематически изображается в виде графа, в котором каждый узел является вершиной. Эта модель представляет собой совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих граф — дерево с иерархической структурой (рис. 1.1). Такой граф имеет единственную вершину, не подчиненную никакой другой вершине и находящуюся на самом верхнем (первом) уровне. Число вершин первого уровня определяет число деревьев в базе данных.

Сетевая модель данных. Эта модель использует ту же терминологию, что и иерархическая модель: «узел», «уровень» и «связь». Единственное отличие между иерархической и сетевой моделями данных заключается в том, что в последней каждый элемент данных (узел) может быть связан с любым другим элементом (уз-

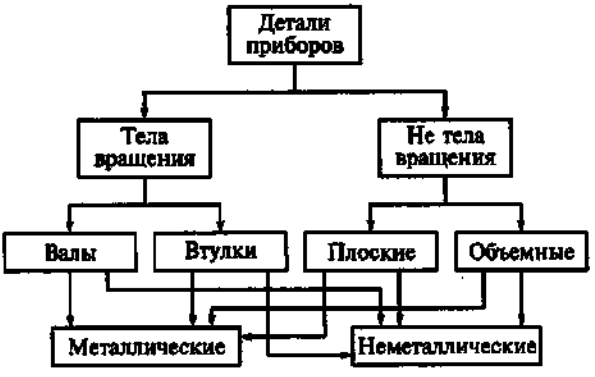


Рис. 1.2. Сетевая модель данных

Таблица 1.1

Детали приборов

Код	Расположение поверхностей	Дополнительная характеристика
1	Тела вращения	Валы
2	Тела вращения	Втулки
3	Не тела вращения	Плоские
4	Не тела вращения	Объемные

лом) (рис. 1.2). Как известно из теории графов, сетевой граф может быть преобразован в граф-дерево.

Реляционная модель данных. Основная идея реляционной модели данных заключается в том, чтобы представить любой набор данных в виде двумерного массива — таблицы. В простейшем случае реляционная модель описывает единственную двумерную таблицу (табл. 1.1), но чаще всего эта модель описывает структуру и взаимоотношения между несколькими различными таблицами. На рис. 1.3 показано разделение табл. 1.1 на две связанные таблицы.

Код	Расположение поверхностей
1	Тела вращения
2	Не тела вращения

Код	Дополнительная характеристика
1.1	Валы
1.2	Втулки
2.1	Плоские
2.2	Объемные

Рис. 1.3. Две связанные таблицы реляционной модели данных

Реляционные модели данных, или реляционные базы данных, являются в настоящее время основным способом в проектировании и организации информационных систем в производстве и бизнесе, поэтому в данном учебном пособии мы рассмотрим теоретические основы и практические методы разработки реляционных баз данных.

1.2. Основы реляционной алгебры

Методы и алгоритмы обработки информации в реляционных базах данных основываются на теории реляционной алгебры, которую ранее называли алгеброй логики, или булевой алгеброй (в дальнейшем мы будем оперировать обоими этим названиями).

Алгебра логики представляет собой прежде всего алгебру высказываний. Под высказыванием в алгебре логики понимают вся-

кое предложение, которое либо истинно, либо ложно; при этом может иметь место только одно из двух указанных значений (например: «Москва — столица России» — истинное высказывание; «снег черен» — ложное высказывание). Отдельные высказывания в алгебре логики обозначаются буквами какого-либо алфавита, например: A, B, C. Истинность или ложность высказываний называется их значениями истинности. В алгебре логики принято отождествлять истинность высказывания с числом 1, а ложность высказывания — с числом 0. Запись A = 1 и C = 0 означает, что A истинно и что C ложно. Каждое конкретное высказывание имеет вполне определенное значение истинности — это постоянная, равная 0 или 1. От конкретных (постоянных) высказываний следует отличать так называемые переменные высказывания.

Переменное высказывание — это не высказывание в подлинном смысле, а переменная для высказываний (пропозициональная переменная), т.е. символ, вместо которого можно подставить постоянные высказывания (или их наименования) и который может принимать лишь два значения: «истинно» или «ложно», или соответственно 1 и 0 (двоичная переменная). Переменные высказывания (т.е. пропозициональные переменные) обозначаются буквами, отличными от тех букв, которыми обозначаются постоянные высказывания. Применение переменных высказываний в алгебре логики служит для выражения всеобщности. Оно позволяет формулировать законы алгебры логики для любых высказываний.

Предметом изучения в алгебре логики являются бинарные (или двузначные) функции, т.е. функции, которые принимают лишь два значения («истинно» или «ложно»; 0 или 1) и зависят от одной или нескольких бинарных переменных. Это так называемые булевы функции.

Из одного или нескольких высказываний, принимаемых за простые, можно составлять сложные высказывания, которые будут бинарными функциями простых высказываний. Объединение простых высказываний в сложные в алгебре логики производится без учета внутреннего содержания (смысла) этих высказываний. Используются определенные логические операции (или логические связи), позволяющие объединять некоторые данные высказывания (постоянные или переменные) в более сложные (постоянные или переменные).

Таблица 1.2

Значения истинности операции отрицания

A	$\bar{A}$
1	0
0	1

К числу основных логических операций относятся следующие: отрицание, конъюнкция, дизъюнкция, эквивалентность, импликация. Логические операции задаются таблично, как функции простых высказываний. В табл. 1.2, 1.3 показаны соответственно таблицы значений для операций отрицания и конъюнкции.

Таблица 1.3

Значения истинности операции конъюнкции

A	B	A ∧ B
1	1	1
0	1	0
1	0	0
0	0	0

Таблица 1.4

Значения истинности операции дизъюнкции

A	B	A ∨ B
1	1	1
0	1	1
1	0	1
0	0	0

Отрицание высказывания. Отрицание высказывания A — это высказывание, которое истинно, когда A ложно, и ложно, когда A истинно; обозначается: $\overline{A}$; читается: «не A».

Конъюнкция двух высказываний. Конъюнкция двух высказываний — это сложное высказывание, которое истинно в случае истинности обоих высказываний, его образующих, и ложно в остальных случаях. Конъюнкция двух высказываний обозначается: A ∧ B; читается «A и B». Знак конъюнкции «∧» имеет смысл союза «и». Операция конъюнкции имеет также смысл логического умножения и может обозначается знаком «&».

Дизъюнкция двух высказываний. Дизъюнкция двух высказываний — это сложное высказывание, которое ложно в случае ложности обоих составляющих его высказываний и истинно в остальных случаях.

Операция дизъюнкции обозначается: A ∨ B; читается: «A или B» (другое обозначение: A + B; другое название — «логическое сложение»).

Знак логической связи «∨» имеет смысл союза «или» и называется знаком дизъюнкции. Союз «или» может употребляться в нескольких различных смыслах. Знак «∨» может иметь смысл «или», употребленный, например, в фразе: «При звоне будильника Петя или Ваня проснется» (здесь «или» не исключает возможности того, что проснутся оба). В данном случае дизъюнкция имеет смысл неразделительного «или». Существует еще исключающее значение союза «или» (например: «Выбирай: он или я»), которое тоже может быть принято за один из видов логических операций, но его не следует смешивать с дизъюнкцией. Дизъюнкция задается табл. 1.4.

Эквивалентность двух высказываний. Эквивалентность двух высказываний — это сложное высказывание, истинное тогда, когда значения истинности составляющих высказываний одинаковы, и ложное — в противном случае; обозначается: A ≡ B; читается: «A эквивалентно B». Для эквивалентности справедливы высказывания,

Таблица 1.5

Значения истинности операции эквивалентности

A	B	A ≡ B
1	1	1
0	1	0
1	0	0
0	0	1

Таблица 1.6

Значения истинности операции отрицания эквивалентности

A	B	$\overline{A \equiv B}$
1	1	0
0	1	1
1	0	1
0	0	0

которые можно записать следующим образом: A ≡ 1 = A, что означает: A эквивалентно единице, когда A истинно.

Запись A ≡ 0 = A означает, что A эквивалентно нулю, когда A ложно. В табл. 1.5 показаны значения истинности операции эквивалентности.

Отрицание эквивалентности. Применив операцию отрицания к высказыванию, представляющему эквивалентность двух высказываний, получим новое сложное высказывание $\overline{A \equiv B}$, называющееся отрицанием эквивалентности. Отрицание эквивалентности означает, что A не эквивалентно B. Отрицание эквивалентности задается табл. 1.6.

Эта операция имеет важное значение в теории проектирования ЭВМ, так как она представляет собой так называемое сложение двоичных чисел по модулю два.

Импликация двух высказываний. Импликация двух высказываний обозначается: A ⊃ B; читается: «если A, то B». Это такое сложное высказывание, которое ложно только в том случае, когда A истинно, а B ложно. Значения истинности этого выражения представлены в табл. 1.7.

Импликация не предполагает обязательную связь по смыслу между условием A и следствием B, хотя и не исключает такую связь. Смысл импликации можно выразить следующим образом: «A ложно или B истинно». В этом выражении союз «или» имеет не исключающее значение.

Таблица 1.7

Значения истинности операции импликации

A	B	A ⊃ B
1	1	1
0	1	1
1	0	0
0	0	1

Любое сложное выражение, полученное из простых высказываний посредством указанных выше логических операций, называется *формулой алгебры логики*. Две формулы алгебры логики, образованные из про-

стых высказываний A_1, A_2, ..., A_n, называются равносильными в том случае, если для каждой комбинации значений истинности высказываний A_1, A_2, ..., A_n обе формулы алгебры логики будут иметь одинаковые значения истинности. Так как существует в точности 2^n различных комбинаций значений истинности n простых высказываний и для каждой из этих комбинаций 2^n сложное выражение будет либо истинным, либо ложным, то может быть 2^{2^n} различных функций алгебры логики, построенных из n данных простых высказываний. В частном случае при двух двоичных переменных A, B (т.е. двух переменных высказываний) можно построить 16 различных функций алгебры логики, т.е. составить 16 неравносильных друг другу сложных логических выражений. Среди этих выражений содержатся все описанные выше логические связи (исключая отрицание, являющееся функцией одной переменной).

Важнейшую роль в алгебре логики играют следующие равносильные соотношения, выражающие основные законы алгебры логики:

1) $\overline{\overline{A}} = A$ означает, что не А истинно, если А ложно;

2) $A \wedge B = B \wedge A$ означает, что (А и В = В и А);

3) $(A \wedge B) \wedge C = A \wedge (B \wedge C)$ означает, что ((А и В) и С) = = (А и (В и С)) или, иначе, это закон логического умножения $((A \cdot B) \cdot C) = (A \cdot (B \cdot C))$;

4) $A \vee B = B \vee A$ означает, что $A + B = B + A$ — известный из курса арифметики закон: «от перемены мест слагаемых сумма не меняется»;

5) $(A \vee B) \vee C = A \vee (B \vee C)$ или $(A + B) + C = A + (B + C)$;

6) $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$ означает, что левая часть выражения истинна при истинности правой части;

7) $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$;

8) $(\overline{A} \vee \overline{B}) = \overline{A \wedge B}$;

9) $(\overline{A} \wedge \overline{B}) = \overline{A \vee B}$;

10) $A \wedge A = A$;

11) $A \vee A = A$;

12) $A \wedge 1 = A$;

13) $A \vee 0 = A$.

Проверка справедливости указанных соотношений может быть произведена на основании определений и таблиц логических операций — конъюнкции, дизъюнкции и отрицания. Соотношения 1—13 используются для преобразования сложных логических выражений к более простому виду. Соотношения 2—5 показывают, что для операций конъюнкции и дизъюнкции справедливы переместительный и сочетательный законы, в силу чего многочленные конъюнкции и дизъюнкции можно записывать без скобок. Например, вместо $[(A \wedge B) \wedge C] \wedge D$ можно записать $A \wedge B \wedge C \wedge D$. Для дальнейшего уменьшения числа скобок в логических формулах при-

няли считать связь с помощью знака «∧» более тесной, чем с помощью знака «∨», а последнюю более тесной, чем связь с помощью знаков «≡», «≡» и «⊃». Выражения вида $A \wedge B \wedge C \wedge D$... часто называются произведением, а его члены — множителями.

Выражения вида $A \vee B \vee C \vee$ ... называют суммой, а его члены — слагаемыми. Соотношение 6 показывает, что в алгебре логики справедлив закон распределительности конъюнкции относительно дизъюнкции. Запись его в виде $A \cdot (B + C) = A \cdot B + A \cdot C$ (где точка означает логическое умножение, а знак «плюс» — логическое сложение) наглядно показывает аналогию между этим законом и законом распределительности умножения относительно сложения в обычной арифметике. Но в отличие от арифметики в алгебре логики имеет место еще закон распределительности дизъюнкции относительно конъюнкции, выражаемый соотношением 7. Оба распределительных закона позволяют производить над формулами алгебры логики преобразования раскрытия скобок и вынесения множителей (а также вынесение общих слагаемых) подобно тому, как это делается в обычной алгебре.

Соотношения 8 и 9 называются законами де Моргана и вместе с соотношением 1 позволяют преобразовывать логические выражения к виду, в котором знаки отрицания относятся только к простым высказываниям.

Помимо соотношений 1—13 весьма полезными для преобразования логических выражений являются следующие равносильные соотношения:

14) $A \vee A \wedge \overline{B} = A \vee B$;

15) $A \wedge (A \vee B) = \overline{A} \vee B$;

16) $A \vee A \wedge B = A$;

17) $A \wedge B \vee A \wedge \overline{C} = A \wedge B \vee A \wedge \overline{C} \wedge B \wedge C$;

18) $A \wedge (A \vee B) = A$;

19) $(A \vee B) \wedge (A \vee \overline{C}) = (A \vee B) \wedge (A \vee \overline{C}) \wedge (B \vee C)$;

20) $A \vee A \wedge \overline{B} = A \vee \overline{B}$;

21) $A \vee (\overline{A} \vee B) = \overline{A} \vee B$;

22) $A \supset B = A \vee \overline{B}$;

23) $A \equiv B = A \wedge B \vee \overline{A} \wedge \overline{B}$.

Использование последних двух соотношений позволяет приводить любые выражения, содержащие знаки «⊃» и «≡» к выражениям, содержащим только знаки «∨», «∧», «¯».

Если под переменными A, B, C, ... будем понимать не только высказывания, но и любую систему элементов, для которой определены действия сложения, умножения и отрицания и которая удовлетворяет соотношениям 1—13, то получим абстрактную алгебру, называемую алгеброй Буля. В частности, высказывания и

основные логические операции «∨», «∧», «¯» представляют собой частный случай, или интерпретацию, алгебры Буля. Другим примером алгебры Буля является алгебра классов, которая дает наглядное геометрическое истолкование для основных логических операций.

Рассмотрим высказывание А, в котором идет речь о принадлежности некоторого свойства *а* предметам какой-то области. Представим себе, что предметы этой области изображаются точками части плоскости, ограниченной некоторым квадратом (рис. 1.4), которую мы обозначим через Q. Ясно, что точки плоскости Q разбиваются на два класса (множества): на класс точек, имеющих свойство *а*, т.е. таких, для которых А = 1, и на класс точек, не имеющих этого свойства, т.е. таких, для которых А = 0, причем каждая точка плоскости Q обязательно принадлежит только одному из этих классов.

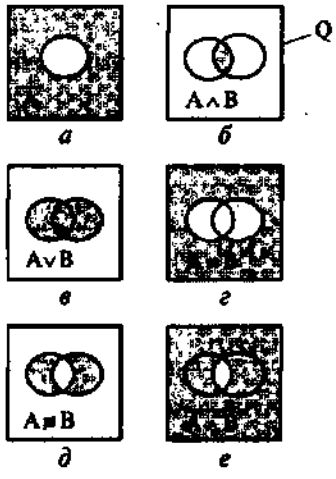


Рис. 1.4. Графическое изображение булевых функций: *а* — множество А; *б* — пересечение множеств; *в* — объединение множеств; *г* — эквивалентности двух множеств; *д* — отрицание эквивалентности; *е* — отрицание конъюнкции

Первый класс можно считать геометрическим изображением высказывания А, или множеством А. При этом может получиться, например, картина, приведенная на рис. 1.4, *а*, на котором высказывание А изображено в виде некоторой области, ограниченной замкнутым (залитым) контуром. Очевидно, что высказывание А (не А) будет тогда изображаться множеством всех остальных точек квадрата Q. Рассмотрим, как при такой интерпретации можно представить основные логические операции.

Конъюнкция двух высказываний будет представляться пересечением двух множеств (рис. 1.4, *б*). Действительно, А ∧ В = 1 только тогда, когда А = 1 и В = 1, а это имеет место лишь для точек, одновременно принадлежащих множеству А и множеству В (их пересечению).

Дизъюнкция двух высказываний А ∨ В будет изображаться множеством, которое получается путем объединения множеств А и В (рис. 1.4, *в*).

Эквивалентность двух высказываний А ≡ В изобразится так, как показано на рис. 1.4, *г*, так как истинность А ≡ В равна 1 либо при А = 1 и В = 1, либо при А = 0 и В = 0.

Отрицание эквивалентности А ≡ В, т.е. высказывание А ≠ В, получается, если учесть, что [illegible] А ≡ В, равно А ≠ В.

Таблица 1.8

Варианты обозначения логических операций

Конъюнкция (логическое умножение)	Дизъюнкция (логическое сложение)	Отрицание (логическое отрицание)	Эквивалентность	Импликация
И	ИЛИ	НЕ		Если... то
AND	OR	NOT		If... then
&	|	¬	→	≡
∧	∨	$\overline{A}$	⊃	↔

Отрицание конъюнкции $\overline{A \wedge B}$ показано на рис. 1.4, *е*.

Подобные диаграммы, называемые диаграммами Венна, могут быть использованы для наглядного представления логических формул с целью их анализа и упрощения.

Рассмотренные логические операции — конъюнкция, дизъюнкция, отрицание — являются независимыми и могут быть выражены друг через друга. В частности, из них можно выделить системы логических операций и с их помощью можно представить все функции алгебры логики. Такие системы логических операций (иногда вместе с константами 1 или 0) называются функционально полными.

В табл. 1.8 представлены различные варианты обозначения логических операций.

Кроме перечисленных выше основных операций алгебры логики существуют еще две операции: отрицание конъюнкции и отрицание дизъюнкции, утверждающие несовместимость соответствующих высказываний.

Отрицание конъюнкции $\overline{A \wedge B}$ обозначается: A / B и называется операцией Шеффера.

Отрицание дизъюнкции $\overline{A \vee B}$, выраженное штрихом Шеффера, имеет смысл $\overline{A} = A / B$.

Операция Шеффера играет важную роль в теории проектирования логических схем процессоров ЭВМ, поскольку электронная схема, реализующая операцию Шеффера, является универсальным функциональным элементом, при помощи которого могут быть построены любые функциональные логические устройства. Графическое представление операции Шеффера приведено на рис. 1.4, *е*.

Операции конъюнкции и дизъюнкции называются двойственными, и формулы алгебры логики, получаемые одна из другой заменой ∧ на ∨ и ∨ на ∧, также называются двойственными. Для двойственных формул F и F* справедлива равносильность высказываний

$$F(A_1, A_2, \ldots, A_n) = \overline{F^*}(\overline{A_1}, \overline{A_2}, \ldots, \overline{A_n}). \qquad (1.1)$$

В алгебре логики устанавливается следующий принцип двойственности: если формулы F и Ф равносильны, то двойственные им формулы F^* и $Ф^*$ также равносильны.

Наиболее наглядно структура формул алгебры логики видна тогда, когда они приведены к одной из двух так называемых нормальных форм. Первая нормальная форма — конъюнктивная (КНФ) — представляет собой некоторую конъюнкцию дизъюнкций, причем в каждой дизъюнкции отдельные члены представляют собой либо простые высказывания (т.е. высказывания, которые не включают в себя других высказываний), либо отрицания простых высказываний. Вторая нормальная форма — дизъюнктивная (ДНФ) — представляет собой некоторую дизъюнкцию конъюнкций, где в каждой конъюнкции отдельные члены являются либо простыми высказываниями, либо их отрицаниями.

Нормальные формы удобны для выделения двух важных классов формул: постоянно-истинных и постоянно-ложных.

Постоянно-истинные формулы всегда равны 1 (совпадают с 1).

Постоянно-ложные формулы всегда равны 0 (совпадают с 0).

Эти классы формул играют существенную роль при упрощении логических выражений.

При упрощении сложных формул, используя равносильности высказываний $A \wedge 1 = A$ и $A \vee 0 = A$, можно отбрасывать постоянно-истинные и постоянно-ложные высказывания, а используя равносильности $A \wedge 0 = 0$ и $A \vee 1 = 1$, можно отбрасывать высказывания, конъюнктивно присоединенные к постоянно-ложному высказыванию и дизъюнктивно присоединенные к постоянно-истинному высказыванию.

Суждение о постоянной истинности сложной формулы может быть получено на основе применения следующих правил:

1) формула $\overline{A} \vee A$ постоянно-истинная;

2) формула $A \vee B$ истинная, если A истинно, а B — произвольное высказывание;

3) формула $A \wedge B$ истинная, если A и B истинны.

Применение этих правил позволяет вывести следующий критерий постоянной истинности сложной формулы.

Постоянно-истинными являются такие формулы, в КНФ которых в каждую дизъюнкцию входит по меньшей мере одно основное высказывание вместе со своим отрицанием. Действительно, в каждой дизъюнкции в этом случае будет, по меньшей мере, один истинный член, а значит, истинны и все дизъюнкции, являющиеся членами КНФ, т.е. будет истинна вся КНФ, представляющая данную формулу алгебры логики.

Аналогичным образом посредством приведения к ДНФ можно определить, является ли заданная формула алгебры логики по-

стоянно-ложной или нет. Формула будет постоянно-ложной, если в каждой из конъюнкций, дизъюнктивно соединенных в ДНФ этой формулы, имеется по крайней мере одно высказывание вместе со своим отрицанием.

Формулы алгебры логики, являющиеся истинными при некоторых значениях входящих в них переменных, называются выполнимыми. Выполнимой будет любая формула, не являющаяся постоянно-ложной.

Существует универсальный способ представления любой функции алгебры логики $F(A_1, A_2, ..., A_n)$ в виде дизъюнкции всех конъюнкций вида

$$F(a_1, a_2, ..., a_n) \wedge A_1' \wedge A_2' \wedge ... \wedge A_n',$$

где $a_1, a_2, ..., a_n$ — набор из значений 0 и 1, а $A_n' = A_n$ при $a_n = 1$; $A_n' = \bar{A}_n$ при $a_n = 0$.

Действительно, для любого набора $a_1, a_2, ..., a_n$ найдется только одна конъюнкция вида (1.1), в которой постоянный множитель $F(a_1, a_2, ..., a_n)$ будет иметь то же значение истинности, что и данная функция при данном наборе, а остальные множители $A_1' \wedge A_2' \wedge ... \wedge A_n'$ будут равны 1. В этой конъюнкции распределение знаков отрицаний над переменными A_n будет совпадать с распределением нулей в наборе $a_1, a_2, ..., a_n$. Оставляя в дизъюнкции только те конъюнкции, у которых постоянные множители равны единице, и удаляя их из конъюнкций по правилу $A \wedge 1 = A$, получим формулу, выражающую данную функцию в форме дизъюнкции конъюнкций вида $A_1' \wedge A_2' \wedge ... \wedge A_n'$. Эта форма называется дизъюнктивной совершенной нормальной формой (ДСНФ) и обладает следующими свойствами:

- не имеет одинаковых слагаемых;
- каждое слагаемое содержит в качестве множителей либо основные переменные, либо их отрицания;
- ни в одном слагаемом нет двух одинаковых множителей и не содержится переменная вместе с ее отрицанием.

Подобным же образом определяется конъюнктивная совершенная нормальная форма (КСНФ), представляющая собой конъюнкцию дизъюнкций, удовлетворяющих аналогичным условиям.

Совершенные нормальные формы могут использоваться при решении вопросов о равносильности сложных формул алгебры логики: две формулы алгебры логики являются равносильными, если они приводятся к одинаковым совершенным нормальным формам. Практическое применение этих форм затрудняется в связи с их громоздкостью, поэтому на практике часто используются так называемые минимальные нормальные формы.

Минимальная дизъюнктивная нормальная форма (МДНФ) представляет собой дизъюнкцию конъюнкций, в которой:

- нет повторяющихся множителей ни в одном слагаемом,

• нет таких пар слагаемых, в которых были бы одинаковые множители;

• для всяких двух слагаемых, в которых имеется одна общая переменная, входящая в одно слагаемое в прямом виде, а в другое слагаемое в виде отрицания, имеется третье слагаемое, представляющее собой конъюнкцию остальных множителей первых двух слагаемых.

Важной областью применения алгебры логики является алгоритмизация процессов переработки информации и управления в реляционных базах данных.

Контрольные вопросы

1. Дайте определение понятий «база данных» и «система управления базами данных».
2. Дайте характеристики иерархических, сетевых и реляционных моделей организации баз данных.
3. Что такое *алгебра логики*?
4. Дайте определения основных логических операций: отрицание, конъюнкция, дизъюнкция, эквивалентность, импликация.
5. Составьте таблицы истинности для каждой из основных логических операций.
6. Что означают или как еще можно записать следующие соотношения:

$A \wedge B = B \wedge A$;
$(A \wedge B) \wedge C = A \wedge (B \wedge C)$;
$A \vee B = B \vee A$;
$(A \vee B) \vee C = A \vee (B \vee C)$;
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$.

7. Допишите правые части следующих законов алгебры логики:

$A \vee (B \wedge C) =$
$\overline{A \vee B} =$
$\overline{A \wedge B} =$
$A \wedge A =$
$A \vee A =$
$A \wedge 1 =$
$A \vee 0 =$

8. Что устанавливают законы де Моргана?
9. Перечислите варианты обозначения логических операций: конъюнкция, дизъюнкция, отрицание, эквивалентность, импликация.
10. Что называется операцией Шеффера, или штрихом Шеффера?

Глава 2
РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ

2.1. Термины и определения

Развитие реляционных баз данных началось в конце 1960-х гг., когда появились первые работы, в которых обсуждались возможности использования привычных для специалиста способов формализованного представления данных в виде таблиц. Некоторые специалисты такой способ представления информации называли таблицами решений, другие — табличными алгоритмами. Теоретики реляционных баз данных табличный способ представления информации называли даталогическими моделями.

Основоположником теории реляционных баз данных считается сотрудник фирмы IBM доктор Э. Ф. Кодд, опубликовавший 6 июня 1970 г. статью «Реляционная модель данных для больших коллективных банков данных» («A Relational Model of Data for Large Shared Data Banks». В этой статье впервые и был использован термин «реляционная модель данных», что и положило начало реляционным базам данных.

Теория реляционных баз данных, разработанная в 1970-х гг. в США доктором Э. Ф. Коддом, опиралась на математический аппарат теории множеств. Он доказал, что любой набор данных можно представить в виде двумерных таблиц особого вида, известных в математике как отношения. От английского слова «relation» («отношение») и произошло название «реляционная модель данных». В настоящее время теоретическую основу проектирования баз данных (БД) составляет математический аппарат реляционной алгебры (см. подразд. 1.2).

Таким образом, реляционная БД представляет собой информацию (данные) об объектах, представленную в виде двумерных массивов — таблиц, объединенных определенными связями. База данных может состоять и из одной таблицы. Прежде чем приступить к дальнейшему изучению реляционных баз данных, рассмотрим применяемые в теории и практике термины и определения.

Таблица базы данных — двумерный массив, содержащий информацию об одном классе объектов. В теории реляционной алгебры двумерный массив (таблицу) называют *отношением*.

Таблица состоит из следующих элементов: поле, ячейка, запись (рис. 2.1).

Поле содержит значения одного из признаков, характеризующих объекты БД. Число полей в таблице соответствует числу признаков, характеризующих объекты БД.

Рис. 2.1. Структура таблицы базы данных

Ячейка содержит конкретное значение соответствующего поля (признака одного объекта).

Запись — строка таблицы. Она содержит значения всех признаков, характеризующих один объект. Число записей (строк) соответствует числу объектов, данные о которых содержатся в таблице.

В теории баз данных термину *запись* соответствует понятие *кортеж* — последовательность атрибутов, связанных между собой отношением AND (И). В теории графов *кортеж* означает простую ветвь ориентированного графа — дерева.

В табл. 2.1 приведены термины, применяемые в теории и практике разработки реляционных баз данных.

Одним из важных понятий, необходимых для построения оптимальной структуры реляционных баз данных, является понятие ключа, или ключевого поля.

Ключом считается поле, значения которого однозначно определяют значения всех остальных полей в таблице. Например, поле «Номер паспорта», или «Идентификационный номер налогоплательщика (ИНН)», однозначно определяет характеристики любого физического лица (при составлении соответствующих таблиц баз данных для отделов кадров или бухгалтерии предприятия).

Таблица 2.1

Термины реляционных баз данных

Теория БД	Реляционные СУБД (FoxPro, Microsoft Access)	SQL Server 7.0
Отношение (Relation)	Таблица (Table)	Таблица (Table)
Атрибут (Attribute)	Поле (Field)	Колонка (Column)
Кортеж (Tuple)	Запись (Record)	Строка (Row)

Ключом таблицы может быть не одно, а несколько полей. В этом случае множество полей может быть возможным ключом таблицы только тогда, когда удовлетворяются два независимых от времени условия: уникальность и минимальность. Каждое поле, не входящее в состав первичного ключа, называется не ключевым полем таблицы.

Уникальность ключа означает, что в любой момент времени таблица базы данных не может содержать никакие две различные записи, имеющие одинаковые значения ключевых полей. Выполнение условия уникальности является обязательным.

Условие *минимальности* ключевых полей означает, что только сочетание значений выбранных полей отвечает требованиям уникальности записей таблицы базы данных. Это означает также, что ни одно из входящих в ключ полей не может быть исключено из него без нарушения уникальности.

При формировании ключа таблицы базы данных, состоящего из нескольких полей, необходимо руководствоваться следующими положениями:

- не следует включать в состав ключа поля таблицы, значения которых сами по себе однозначно идентифицируют записи в таблице. Например, не стоит создавать ключ, содержащий одновременно поля «номер паспорта» и «идентификационный номер налогоплательщика», поскольку каждый из этих атрибутов может однозначно идентифицировать записи в таблице;
- нельзя включать в состав ключа неуникальное поле, т. е. поле, значения которого могут повторяться в таблице.

Каждая таблица должна иметь, по крайней мере, один возможный ключ, который выбирается в качестве *первичного ключа*. Если в таблице существуют поля, значения каждого из которых однозначно определяют записи, то эти поля могут быть приняты в качестве *альтернативных ключей*. Например, если в качестве первичного ключа выбрать идентификационный номер налогоплательщика, то номер паспорта будет альтернативным ключом.

2.2. Нормализация таблиц реляционной базы данных

Реляционная база данных представляет собой некоторое множество таблиц, связанных между собой. Число таблиц в одном файле или одной базе данных зависит от многих факторов, основными из которых являются:

- состав пользователей базы данных,
- обеспечение целостности информации (особенно важно в многопользовательских информационных системах),
- обеспечение наименьшего объем требуемой памяти и минимального времени обработки данных.

Учет данных факторов при проектировании реляционных баз данных осуществляется методами нормализации таблиц и установлением связей между ними.

Нормализация таблиц представляет собой способы разделения одной таблицы базы данных на несколько таблиц, в целом отвечающих перечисленным выше требованиям.

Нормализация таблицы представляет собой последовательное изменение структуры таблицы до тех пор, пока она не будет удовлетворять требованиям последней формы нормализации. Всего существует шесть форм нормализации:

- первая нормальная форма (First Normal Form — 1NF);
- вторая нормальная форма (Second Normal Form — 2NF);
- третья нормальная форма (Third Normal Form — 3NF);
- нормальная форма Бойса — Кодда (Brice — Codd Normal Form — BCNF);
- четвертая нормальная форма (Fourth Normal Form — 4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (Fifth Normal Form — 5NF, или PJ/NF).

При описании нормальных форм используются следующие понятия: «функциональная зависимость между полями»; «полная функциональная зависимость между полями»; «многозначная функциональная зависимость между полями»; «транзитивная функциональная зависимость между полями»; «взаимная независимость между полями».

Функциональной зависимостью между полями A и B называется зависимость, при которой каждому значению A в любой момент времени соответствует единственное значение B из всех возможных. Примером функциональной зависимости может служить связь между идентификационным номером налогоплательщика и номером его паспорта.

Полной функциональной зависимостью между составным полем A и полем B называется зависимость, при которой поле B зависит функционально от поля A и не зависит функционально от любого подмножества поля A.

Многозначная функциональная зависимость между полями определяется следующим образом. Поле A многозначно определяет поле B, если для каждого значения поля A существует «хорошо определенное множество» соответствующих значений поля B. Например, если рассматривать таблицу успеваемости учащихся в школе, включающую в себя поля «Предмет» (поле A) и «Оценка» (поле B), то поле B имеет «хорошо определенное множество» допустимых значений: 1, 2, 3, 4, 5, т.е. для каждого значения поля «Предмет» существует многозначное «хорошо определенное множество» значений поля «Оценка».

Транзитивная функциональная зависимость между полями A и C существует в том случае, если поле C функционально зависит от

поля B, а поле B функционально зависит от поля A; при этом не существует функциональной зависимости поля A от поля B.

Взаимная независимость между полями определяется следующим образом. Несколько полей взаимно независимы, если ни одно из них не является функционально зависимым от другого.

Первая нормальная форма. Таблица находится в первой нормальной форме тогда и только тогда, когда ни одно из полей не содержит более одного значения и любое ключевое поле не пусто.

Первая нормальная форма является основой реляционной модели данных. Любая таблица в реляционной базе данных автоматически находится в первой нормальной форме, иное просто невозможно по определению. В такой таблице не должно содержаться полей (признаков), которые можно было бы разделить на несколько полей (признаков).

Ненормализованными, как правило, бывают таблицы, изначально не предназначенные для компьютерной обработки содержащейся в них информации. Например, в табл. 2.2 показан фрагмент таблицы из справочника «Универсальные металлорежущие станки», изданного Экспериментальным научно-исследовательским институтом металлорежущих станков (ЭНИМС).

Данная таблица является ненормализованной по следующим причинам.

1. Она содержит строки, имеющие в одной ячейке несколько значений одного поля: «Наибольший диаметр обработки, мм» и «Частота вращения шпинделя, об/мин».

2. Одно поле — «Габаритные размеры (длина × ширина × высота), мм» может быть разделено на три поля: «Длина, мм», «Ширина, мм» и «Высота, мм». Целесообразность такого разделения может быть обоснована необходимостью последующих расчетов площадей или занимаемых объемов.

Исходная таблица должна быть преобразована в первую нормальную форму. Для этого необходимо:

• поля «Наибольший диаметр обработки, мм» и «Частота вращения шпинделя, об/мин» разделить на несколько полей в соответствии с числом значений, содержащихся в одной ячейке;

Таблица 2.2

Станки токарной группы

№ п/п	Модель станка	Наибольший диаметр обработки, мм	Частота вращения шпинделя, об/мин	Габаритные размеры (длина × × ширина × высота), мм
1	1Д12	12 (восьми — шестигранник)	112...5000 (левое); 56...630 (правое)	1630×740×1410

Таблица 2.3

Нормальная форма табл. 2.2

№ п/п	Модель станка	Наибольший диаметр цилиндрической заготовки, мм	Наибольший диаметр нецилиндрической заготовки, мм	Диапазон частоты вращения шпинделя (левое), об/мин	Диапазон частоты вращения шпинделя (правое), об/мин	Длина, мм	Ширина, мм	Высота, мм
1	1Д12	12	8	112... 5000	56... 630	163	740	141

• поле «Габаритные размеры (длина × ширина × высота), мм» разделить на три поля: «Длина, мм», «Ширина, мм», «Высота, мм».

Ключевым полем данной таблицы может быть поле «Модель станка» или «№ п/п»

Вид нормальной формы имеет табл. 2.3.

Рассмотрим еще один пример. На рис. 2.2 показан фрагмент бланка зачетно-экзаменационной ведомости, который, как и в предыдущем примере, изначально не предназначался для компьютерной обработки.

Пусть мы хотим создать базу данных для автоматизированной обработки результатов зачетно-экзаменационной сессии в соот-

Факультет № ...					Группа ...					
Семестр ...					По дисциплине ...					
Экзаменатор ...										
№ п/п	ФИО	Экзамен 1 Зачет			Экзамен 2 Зачет			Экзамен 3 Зачет		
		Дата, подпись	Дата, подпись	Дата, подпись	Дата, подпись	Дата, подпись	Дата, подпись	Дата, подпись	Дата, подпись	Дата, подпись
...	...	...	...	...	...	...	...	...	...	...
Подпись декана факультета _____					Подпись экзаменатора _____					

Рис. 2.2. Бланк зачетно-экзаменационной ведомости

ветствии с содержанием зачетно-экзаменационной ведомости. Для этого преобразуем содержание бланка в таблицы базы данных. Исходя из необходимости соблюдения условий функциональной зависимости между полями необходимо сформировать, как минимум, две таблицы (рис. 2.3) (ключевые поля в каждой таблице выделены полужирным шрифтом). В первой таблице содержатся результаты сдачи зачета (экзамена) каждым студентом по конкретному предмету. Во второй таблице содержатся результирующие итоги сдачи зачета (экзамена) конкретной группы студентов по конкретному предмету. В первой таблице ключевым является поле «ФИО студента», а во второй таблице — поле «Дисциплина». Таблицы должны быть связаны между собой по полям «Дисциплина» и «Шифр группы».

Представленные структуры таблиц полностью отвечает требованиям первой нормальной формы, но характеризуется следующими недостатками:

- добавление новых данных в таблицы требует ввода значений для всех полей;
- в каждую строку каждой таблицы необходимо вводить повторяющиеся значения полей «Дисциплина», «ФИО преподавателя», «Шифр группы».

Следовательно, при таком составе таблиц и их структуре имеется явная избыточность информации, что, естественно, потребует дополнительных объемов памяти.

Чтобы избежать перечисленных недостатков, необходимо привести таблицы ко второй или третьей нормальной форме.

Вторая нормальная форма. Таблица находится во второй нормальной форме, если она удовлетворяет требованиям первой нормальной формы и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом.

Дисциплина	**ФИО преподавателя**	**Шифр группы**	**ФИО студента**	**№ по журналу**	**Дата сдачи**	**Оценка**	**Подпись преподавателя**

Дисциплина	**ФИО преподавателя**	**Шифр группы**	**Количество отл.(5)**	**Количество хор.(4)**	**Количество уд.(3)**	**Количество неуд.(2)**	**Средняя успеваемость**

Рис. 2.3. Исходные таблицы зачетно-экзаменационной ведомости

Если таблица имеет простой первичный ключ, состоящий только из одного поля, то она автоматически находится во второй нормальной форме.

Если же первичный ключ составной, то таблица необязательно находится во второй нормальной форме. Тогда ее необходимо разделить на две или более таблиц таким образом, чтобы первичный ключ однозначно идентифицировал значение в любом поле. Если в таблице имеется хотя бы одно поле, не зависящее от первичного ключа, то в первичный ключ необходимо включить дополнительные колонки. Если таких колонок нет, то необходимо добавить новую колонку.

Исходя из данных условий, определяющих вторую нормальную форму, можно сделать следующие выводы по характеристике составленных таблиц (см. рис. 2.3).

В первой таблице нет прямой связи между ключевым полем и полем «ФИО преподавателя», поскольку зачет или экзамен по одному предмету могут принимать разные преподаватели. В таблице существует полная функциональная зависимость только между всеми остальными полями и ключевым полем «Дисциплина».

Аналогично во второй таблице нет прямой связи между ключевым полем и полем «ФИО преподавателя».

Для оптимизации базы данных, в частности для уменьшения требуемого объема памяти из-за необходимости повторения в каждой записи значений полей «Дисциплина» и «ФИО преподавателя», необходимо изменить структуру базы данных — преобразовать исходные таблицы во вторую нормальную форму. Состав таблиц измененной структуры базы данных показан на рис. 2.4.

Преобразованная структура базы данных состоит из шести таблиц, две из которых связаны между собой (ключевые поля в каждой таблице выделены полужирным шрифтом). Все таблицы удовлетворяют требованиям второй нормальной формы.

Пятая и шестая таблицы имеют в полях повторяющиеся значения, но, учитывая, что эти значения представляют собой целые числа вместо текстовых данных, общий объем требуемой памяти для хранения информации значительно меньше, чем в исходных таблицах (см. рис. 2.1).

Кроме того, новая структура базы данных обеспечит возможность заполнения таблиц различными специалистами (подразделениями управленческих служб). Дальнейшая оптимизация таблиц баз данных сводится к приведению их к третьей нормальной форме.

Третья нормальная форма. Таблица находится в третьей нормальной форме, если она удовлетворяет определению второй нормальной формы и ни одно из ее не ключевых полей не зависит функционально от любого другого не ключевого поля.

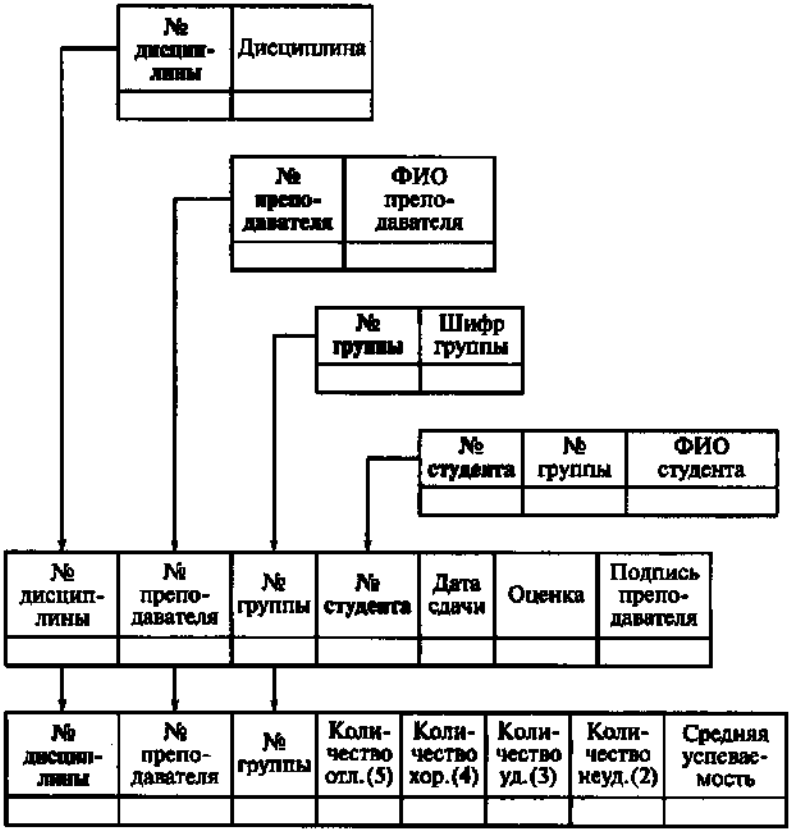


Рис. 2.4. Таблицы зачетно-экзаменационной ведомости, нормализованные до второй формы

Можно также сказать, что таблица находится в третьей нормальной форме, если она находится во второй нормальной форме и каждое не ключевое поле не транзитивно зависит от первичного ключа. Требование третьей нормальной формы сводится к тому, чтобы все не ключевые поля зависели только от первичного ключа и не зависели друг от друга.

В соответствии с этими требованиями в составе таблиц базы данных (см. рис. 2.4) к третьей нормальной форме относятся первая, вторая, третья и четвертая таблицы.

Для приведения пятой и шестой таблиц к третьей нормальной форме создадим новую таблицу, содержащую информацию о составе предметов, по которым проводятся экзамены или зачеты в группах студентов. В качестве ключа создадим поле «Счетчик», устанавливающий номер записи в таблице, так как каждая запись должна быть уникальна.

В результате получим новую структуру базы данных, которая показана на рис. 2.5 (ключевые поля в каждой таблице выделены полужирным шрифтом). В данной структуре содержится семь таблиц, которые отвечают требованиям третьей нормальной формы.

Нормальная форма Бойса—Кодда. Таблица находится в нормальной форме Бойса—Кодда только в том случае, если любая функциональная зависимость между ее полями сводится к полной функциональной зависимости от возможного ключа.

Согласно данному определению в структуре базы данных (см. рис. 2.4) все таблицы соответствуют требованиям нормальной формы Бойса—Кодда.

Дальнейшая оптимизация таблиц баз данных должна сводиться к полной декомпозиции таблиц.

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

Проекцией называют копию таблицы, в которую не включены одна или несколько колонок новой таблицы.

Четвертая нормальная форма. Четвертая нормальная форма является частным случаем пятой нормальной формы, когда полная декомпозиция должна быть соединением двух проекций. Очень

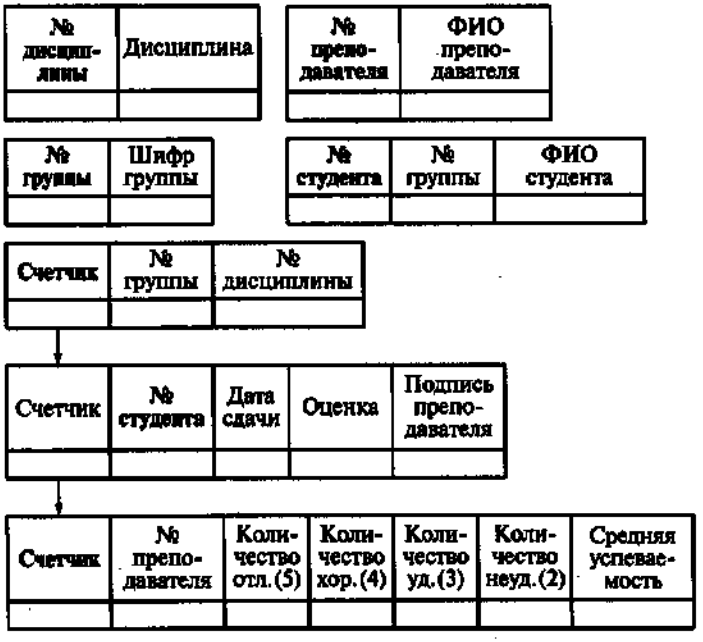


Рис. 2.5. Таблицы зачетно-экзаменационной ведомости, нормализованные до третьей формы

трудно найти такую таблицу, чтобы она находилась в четвертой нормальной форме, но не удовлетворяла определению пятой нормальной формы.

Пятая нормальная форма. Таблица находится в пятой нормальной форме тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в пятой нормальной форме.

На практике оптимизация таблиц базы данных заканчивается третьей нормальной формой. Приведение таблиц к четвертой и пятой нормальным формам представляет, по нашему мнению, чисто теоретический интерес. Практически эта проблема решается разработкой запросов на создание новой таблицы.

2.3. Проектирование связей между таблицами

Процесс нормализации исходных таблиц баз данных позволяет создать оптимальную структуру информационной системы — разработать базу данных, требующую наименьших ресурсов памяти и, как следствие, обеспечивающую наименьшее время доступа к информации.

В то же время разделение одной исходной таблицы на несколько требует выполнения одного из важнейших условий проектирования информационных систем — обеспечения целостности информации в процессе эксплуатации базы данных.

В приведенном выше примере нормализации исходных таблиц (см. рис. 2.3) из двух таблиц в конечном итоге мы получили семь таблиц, приведенных к третьей и четвертой нормальным формам.

Как показывает практика, в реальном производстве и бизнесе базы данных представляют собой многопользовательские системы. Это относится как к созданию и поддержанию данных в отдельных таблицах, так и к использованию информации для принятия решений.

В рассмотренном выше примере, в реально функционирующей системе управления учебным процессом в вузе или колледже, первоначальное формирование учебных групп производится приемными комиссиями при зачислении абитуриентов по результатам вступительных экзаменов. Дальнейшее поддержание информации о составе студентов в группах в вузах возлагается на деканаты, а в колледжах — на учебные отделения или соответствующие структуры. Состав учебных дисциплин по группам определяется другими службами или специалистами. Информация о преподавательском составе формируется в отделах кадров. Результаты зачетных и экзаменационных сессий необходимы руководителям деканата и отделений, в том числе и для принятия решений о предоставле-

связана с несколькими строками подчиненной таблицы; при этом любая строка подчиненной таблицы связана только с одной строкой главной таблицы;

• связь «многие ко многим» устанавливается в случаях, когда конкретная строка главной таблицы в любой момент времени связана с несколькими строками подчиненной таблицы и в то же время одна строка подчиненной таблицы связана с несколькими строками главной таблицы.

При изменении значения первичного ключа в главной таблице возможны следующие варианты поведения зависимой таблицы.

Каскадирование (Cascading). При изменении данных первичного ключа в главной таблице происходит изменение соответствующих данных внешнего ключа в зависимой таблице. Все имеющиеся связи сохраняются.

Ограничение (Restrict). При попытке изменить значение первичного ключа, с которым связаны строки в зависимой таблице, изменения отвергаются. Допускается изменение лишь тех значений первичного ключа, для которых не установлена связь с зависимой таблицей.

Установление (Relation). При изменении данных первичного ключа внешний ключ устанавливается в неопределенное значение (NULL). Информация о принадлежности строк зависимой таблицы теряется. Если изменить несколько значений первичного ключа, то в зависимой таблице образуется несколько групп строк, которые ранее были связаны с измененными ключами. После этого невозможно определить, какая строка с каким первичным ключом была связана.

На рис. 2.6 показаны схемы связей между таблицами базы данных, представленной на рис. 2.5.

Контрольные вопросы

1. Дайте определения следующим элементам таблицы баз данных: поле, ячейка, запись.
2. Что означают понятия «ключ», «ключевое поле»?
3. Какое ключевое поле называют первичным ключом, а какое — внешним ключом?
4. В чем состоит процесс нормализации таблиц базы данных?
5. Какие пять нормальных форм таблиц баз данных вы знаете?
6. Дайте определения следующим типам связей между таблицами базы данных: «один к одному»; «один ко многим»; «многие ко многим».

Глава 3

ИНФОРМАЦИОННЫЕ МОДЕЛИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

3.1. Типы информационных моделей

В предыдущей главе мы рассмотрели теоретические основы формирования оптимального состава таблиц в реляционных базах данных на основе принципов нормализации таблиц. Естественно, что решать любую задачу, в том числе и информационную, можно различными способами. Для оценки вариантов проектируемых баз данных разрабатывают информационные модели. Разработка информационных моделей пока не имеет четких формализованных правил, поэтому эта задача, также как и процесс традиционного программирования, является «искусством» и зависит от квалификации разработчиков. Тем не менее существуют основные принципы создания информационных моделей для оценки оптимальности проектируемых баз данных.

Информационная модель данных предусматривает три уровня описания системы: концептуальный, логический, физический, а соответственно — три типа информационных моделей.

3.2. Концептуальные модели данных

Концептуальный уровень описания базы данных (концептуальная модель) представляет собой информационные объекты и их взаимосвязи без указания способов описания и хранения данных.

В данном определении информационными объектами будем называть классы объектов, сведения о которых хранятся в таблицах базы данных. Как правило, в таблице базы данных содержатся сведения об объектах одного класса.

Классом называют множество объектов, характеризующихся одинаковым набором признаков.

Данные об информационных объектах одного класса могут находиться в одной или нескольких таблицах.

Данные об информационных объектах разных классов должны находиться в разных таблицах.

Конечной задачей разработки концептуальной модели является установление оптимального состава таблиц базы данных.

Изложенные в предыдущих подразделах методы формирования состава таблиц базы данных на основе принципов нормализации в конечном итоге определяют концептуальную модель базы данных.

Так, в примере, изложенном в подразд. 1.3, согласно поставленной задаче была разработана концептуальная модель базы данных, предусматривающая создание семи таблиц.

3.3. Логические модели данных

Логический уровень описания базы данных (логическая модель) отражает логические связи между таблицами. В подразд. 1.4 мы познакомились с правилами формирования связей между таблицами. Показанная на рис. 2.6 схема связей между таблицами и является логической моделью соответствующей базы данных. Между таблицами «Дисциплины», «Группы» и таблицей «Дисциплины по группам» установлены связи «один ко многим». Такой же характер связей установлен между таблицами «Преподаватели» и «Результаты по группам», а также между таблицами «Результаты по группам» и «Результаты по студентам». Между таблицами «Студенты» и «Результаты по студентам» установлена связь «один к одному».

Анализируя данную логическую модель, можно сделать вывод, что данная структура базы данных позволяет обеспечить целостность информации при любых изменениях в таблицах, которые осуществляют различные специалисты управленческих служб учебного заведения.

3.4. Физические модели данных

Физический уровень описания реляционной базы данных (физические модели) характеризуют способы обработки и хранения информации. В теории и практике разработки баз данных и систем управления базами данных разделяют два подхода к построению физических моделей данных.

Первый подход не связан с конкретной СУБД и предполагает описание физических свойств данных каждой таблицы — *физические модели таблиц базы данных.*

Второй подход к разработке физической модели связан с разработкой архитектуры, организации и хранения данных в конкретной СУБД — *физические модели хранения данных.*

Разработчик базы данных может не знать архитектуру прикладной программной системы, с помощью которой он создает свою информационную систему, но при этом он должен проработать физические модели для каждой таблицы.

Физические модели таблиц базы данных. Физическая модель таблицы базы данных предполагает описание свойств каждого поля таблицы. Для описания свойств полей необходимо составить проект таблицы по форме, показанной на рис. 3.1.

№ п/п	Имя поля	Подпись поля	Тип данных	Количество символов	Точность	Ключ (да)

Рис. 3.1. Проект описания структуры таблицы БД

Таким образом, разработка физической модели проекта таблицы базы данных сводится к описанию характеристик каждого поля. Приведем обязательные характеристики полей таблиц базы данных.

Имя поля — некоторый минимальный набор символов, предназначенный для поиска данных в таблице. В каждой прикладной программной системе для разработки баз данных существуют свои грамматические правила для формирования имен полей. В общем случае не допускается начинать имя поля с символа пробела, выбирать в качестве символов знаки препинания.

Подпись поля идентифицируется с названием признака объекта, значения которого будут храниться в ячейках поля. Подпись поля будет находиться в заголовке таблицы. В современных СУБД не существует каких-либо ограничений на формирование подписи поля.

Тип данных — обозначение типа данных в соответствии с конкретной программной системой.

Количество символов — предполагаемое количество символов, которые будут храниться в ячейках поля.

Точность — число знаков после запятой в числовых полях.

Ключ — указание, что данное поле является ключевым.

Данный состав свойств является минимально необходимым для описания данных, хранимых в таблице.

Физические модели хранения данных. Физические модели хранения данных определяют методы размещения данных в памяти компьютера или на соответствующих носителях информации, а также способы хранения и доступа к этим данным. Исторически первыми системами хранения и доступа были файловые структуры и системы управления файлами (СУФ). Фактически файловые структуры хранения информации являлись и являются основой операционных систем. В системах управления базами данных использование файловых систем хранения информации оказалось не эффективным потому, что пользователю требовалась информация в виде отдельных данных, а не содержание всего файла. Поэтому в современных СУБД перешли от файловых структур к непосредственному размещению данных на внешних носителях — устройствах внешней памяти. Однако механизмы управления, применяемые в файловых системах, во многом перешли и в новые системы организации данных во внешней памяти, называемые чаще страничными системами хранения информации. Поэтому

раздел, посвященный физическим моделям данных, мы начнем с обзора файлов и файловых структур.

Файловые структуры организации базы данных. В каждой СУБД по-разному организованы хранение и доступ к данным, однако существуют некоторые файловые структуры, которые применяются практически во всех СУБД.

В системах баз данных файлы и файловые структуры, которые используются для хранения информации во внешней памяти, можно классифицировать (рис. 3.2).

С точки зрения пользователя, *файл* представляет собой поименованную область дискового пространства, в которой хранится некоторая последовательность записей. В таком файле всегда можно определить первую и последнюю запись; текущую запись; запись, предшествующую текущей и следующую за ней.

В соответствии с методами управления доступом к информации в файлах различают устройства внешней памяти (накопители информации) с произвольной адресацией, или прямым доступом (магнитные и оптические диски), и устройства с последовательной адресацией, или последовательным доступом (магнитофоны, стриммеры).

На устройствах с *произвольной адресацией* возможна установка головок для чтения записи в любую область накопителя практически мгновенно.

На устройствах *с последовательной адресацией* вся память рассматривается как линейная последовательность информационных элементов. Поэтому в таких накопителях для получения информации требуется пройти некоторый путь от исходного состояния считывающего устройства до нужной записи.

Файлы с постоянной длиной записи, расположенные на устройствах прямого доступа (УПД), являются *файлами прямого доступа*.

В этих файлах физический адрес расположения нужной записи может быть вычислен по номеру записи (NZ).

Каждая файловая система — система управления файлами — поддерживает некоторую иерархическую файловую структуру,

Рис. 3.2. Классификация файловых структур

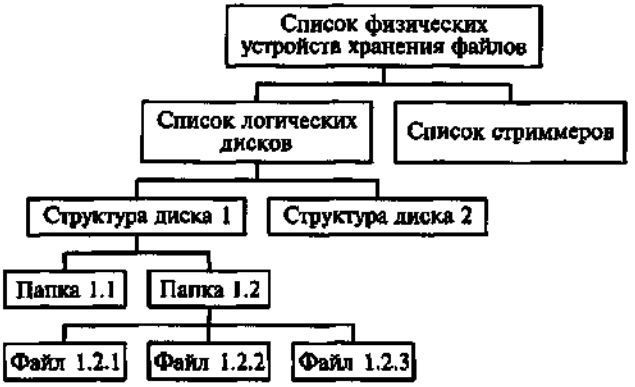


Рис. 3.3. Иерархическая файловая структура

включающую чаще всего ограниченное число уровней иерархии в представлении внешней памяти (рис. 3.3).

Для каждого файла в системе хранится следующая информация:

- имя файла;
- тип файла (например, расширение или другие характеристики);
- размер записи;
- число занятых физических блоков;
- базовый начальный адрес;
- ссылка на сегмент расширения;
- способ доступа (код защиты).

Для файлов с постоянной длиной записи адрес размещения записи с номером K может быть вычислен по формуле

$$BA + (K - 1) \cdot LZ + 1,$$

где BA — базовый адрес; LZ — длина записи.

Если можно определить адрес, на который необходимо позиционировать механизм считывания записи, то устройства прямого доступа делают это практически мгновенно, поэтому для таких файлов чтение произвольной записи практически не зависит от ее номера. Файлы прямого доступа обеспечивают наиболее быстрый доступ к произвольным записям, и их использование считается наиболее перспективным в системах баз данных.

На устройствах последовательного доступа могут быть организованы файлы только последовательного доступа.

Они могут быть организованы двумя способами:

1) конец записи отмечается специальным маркером;

2) в начале каждой записи записывается ее длина.

Файлы с прямым доступом обеспечивают достаточно надежный способ доступа к записи. Основным недостатком файлов пря-

мого доступа является то, что поиск записи производится по ее номеру, что при большом числе записей занимает существенное время. Чаще всего в базах данных целесообразно организовывать поиск записей по ключевым полям. Но в этом случае неизвестен соответствующий номер записи. В подобных случаях применяют различные методы так называемого хэширования (рандомизации).

Суть методов хэширования состоит в том, что выбираются значения ключа (или некоторые его характеристики), которые используются для начала поиска, т.е. вычисляется так называемая хэш-функция $h(k)$, где k — значение ключевого поля. В этом случае число шагов поиска значительно уменьшается. Однако при таком подходе возможны ситуации, когда нескольким разным ключам может соответствовать одно значение хэш-функции, т.е. один адрес. Подобные ситуации называются *коллизиями*. Значения ключей, которые имеют одно и то же значение хэш-функции, называются *синонимами*.

Поэтому при использовании хэширования как метода доступа необходимо принять два независимых решения:

- выбрать хэш-функцию;
- выбрать метод разрешения коллизий.

Существует множество различных стратегий разрешения коллизий, наиболее распространенными из которых являются:

- разрешение коллизии с помощью области переполнения;
- разрешение коллизии методом свободного замещения.

Разрешение коллизии с помощью области переполнения. При выборе этой стратегии область хранения разбивается на две части: основную область и область переполнения.

Для каждой новой записи вычисляется значение хэш-функции, которое определяет адрес ее расположения, и запись заносится в основную область в соответствии с полученным значением хэш-функции.

Если вновь заносимая запись имеет такое же значение функции хэширования, которое использовала другая запись, уже имеющаяся в БД, то новая запись заносится в область переполнения на первое свободное место, а в записи-синониме, которая находится в основной области, делается ссылка на адрес вновь размещенной записи в области переполнения. Если же уже существует ссылка в записи-синониме, которая расположена в основной области, то новая запись получает дополнительную информацию в виде ссылки и уже в таком виде заносится в область переполнения.

При таком алгоритме время размещения любой новой записи составляет не более двух обращений к диску, с учетом того, что номер первой свободной записи в области переполнения хранится в виде системной переменной.

Рассмотрим теперь механизмы поиска произвольной записи и удаления записи для этой стратегии хэширования.

При поиске записи также сначала вычисляется значение ее хэш-функции и считывается первая запись в цепочке синонимов, которая расположена в основной области. Если искомая запись не соответствует первой в цепочке синонимов, то поиск осуществляется перемещением по цепочке синонимов до тех пор, пока не будет обнаружена требуемая запись. Скорость поиска зависит от длины цепочки синонимов, поэтому качество хэш-функции определяется максимальной длиной цепочки синонимов. Хорошим результатом может считаться наличие не боле 10 синонимов в цепочке.

При удалении произвольной записи сначала определяется ее место расположения. Если удаляемой является первая запись в цепочке синонимов, то после удаления на ее место в основной области заносится вторая (следующая) запись в цепочке синонимов; при этом все указатели (ссылки на синонимы) сохраняются.

Если же удаляемая запись находится в середине цепочки синонимов, то необходимо провести корректировку указателей: в записи, предшествующей удаляемой, в цепочке ставится указатель из удаляемой записи. Если это последняя запись в цепочке, то все равно механизм изменения указателей такой же, т.е. в предшествующую запись заносится признак отсутствия следующей записи в цепочке, который ранее хранился в последней записи.

Разрешение коллизии методом свободного замещения. При этой стратегии файловое пространство не разделяется на области, но для каждой записи добавляются два указателя: указатель на предыдущую запись в цепочке синонимов и указатель на следующую запись в цепочке синонимов. Отсутствие соответствующей ссылки обозначается специальным символом, например нулем. Для каждой новой записи вычисляется значение хэш-функции и, если данный адрес свободен, запись попадает на заданное место и становится первой в цепочке синонимов. Если адрес, соответствующий полученному значению хэш-функции, занят, то по наличию ссылок определяется, является ли запись, расположенная по указанному адресу, первой в цепочке синонимов. Если да, то новая запись располагается на первом свободном месте и для нее устанавливаются соответствующие ссылки: она становится второй в цепочке синонимов, на нее ссылается первая запись, а она ссылается на следующую, если такая есть.

Если запись, которая занимает требуемое место, не является первой записью в цепочке синонимов, значит, она занимает данное место «незаконно» и при появлении «законного владельца» должна быть «выселена», т.е. перемещена на новое место. Механизм перемещения аналогичен занесению новой записи, которая уже имеет синоним, занесенный в файл. Для этой записи ищется первое свободное место и корректируются соответствующие ссыл-

ки: в записи, которая является предыдущей в цепочке синонимов для перемещаемой записи, заносится указатель на новое место перемещаемой записи, указатели же в самой перемещаемой записи остаются прежние.

После перемещения «незаконной» записи вновь вносимая запись занимает свое законное место и становится первой записью в новой цепочке синонимов.

Механизмы удаления записей во многом аналогичны механизмам удаления в стратегии с областью переполнения и сводятся к следующим действиям.

Если удаляемая запись является первой записью в цепочке синонимов, то после удаления на ее место перемещается следующая (вторая) запись из цепочки синонимов и проводится соответствующая корректировка указателя третьей записи в цепочке синонимов, если такая существует.

Если же удаляется запись, которая находится в середине цепочки синонимов, то производится только корректировка указателей: в предшествующей записи указатель на удаляемую запись заменяется указателем на следующую за удаляемой запись, а в записи, следующей за удаляемой, указатель на предыдущую запись заменяется на указатель на запись, предшествующую удаляемой.

Индексные файлы. Несмотря на высокую эффективность хэш-адресации в файловых структурах не всегда удается найти соответствующую функцию, поэтому при организации доступа по первичному ключу широко используются индексные файлы.

Индексные файлы можно представить как файлы, состоящие из двух частей. Сначала идет индексная область, которая занимает некоторое целое число блоков, а затем идет основная область, в которой последовательно расположены все записи файла.

В некоторых системах индексными файлами называются также и файлы, организованные в виде инвертированных списков, которые используются для доступа по вторичному ключу. В зависимости от организации индексной и основной областей различают два типа файлов: с плотным индексом (индексно-прямые файлы) и с неплотным индексом (индексно-последовательные файлы).

Файлы с плотным индексом, или индексно-прямые файлы. В этих файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а структура индексной записи в них имеет следующий вид:

Значение ключа	Номер записи

Здесь *значение ключа* — это значение первичного ключа, а *номер записи* — это порядковый номер записи в основной области, которая имеет данное значение первичного ключа.

Так как индексные файлы строятся для первичных ключей, однозначно определяющих запись, то в них не может быть двух записей, имеющих одинаковое значение первичного ключа. В индексных файлах с плотным индексом для каждой записи в основной области существует одна запись из индексной области. Все записи в индексной области упорядочены по значению ключа, поэтому можно применить более эффективные способы поиска в упорядоченном пространстве.

Длина доступа к произвольной записи оценивается не в абсолютных значениях, а числом обращений к устройству внешней памяти, которым обычно является диск. Именно обращение к диску является наиболее длительной операцией по сравнению со всеми обработками в оперативной памяти.

Наиболее эффективным алгоритмом поиска на упорядоченном массиве является логарифмический, или бинарный, поиск. В теории вероятности его называют методом половинного деления. Появление этого метода связано с теорией артиллерийской стрельбы. Для попадания в цель все пространство, на котором находится цель, делят пополам. Затем производят выстрел в ту половину, на которой предположительно находится объект. Устанавливают точку попадания снаряда. Если был перелет, то производят корректировку — область между точкой попадания снаряда и половиной отрезка снова делят на две части — и стреляют еще раз.

В этом случае максимальное число шагов поиска (попадания в цель) определяется двоичным логарифмом от общего числа элементов (целей) в искомом пространстве поиска:

$$T_и = \log_2 N, \qquad (3.1)$$

где N — число элементов.

При поиске записей существенным является только число обращений к диску по заданному значению первичного ключа. Сначала производится поиск в индексной области, где применяется двоичный алгоритм поиска индексной записи, а затем путем прямой адресации в основной области производится поиск по номеру записи. Для того чтобы оценить максимальное время доступа к записи, необходимо определить число обращений к диску в процессе поиска.

В соответствии с формулой (3.1) число обращений к диску при поиске записи определится следующим образом:

$$T_и = \log_2 N_{инд.\ обл} + 1,$$

где $N_{инд.\ обл}$ — число индексных блоков, в которых размещаются все записи.

Учитывая что после поиска записи в индексном блоке нужно еще раз обратиться к основной области, в формуле добавилась единица (+ 1).

Таблица 3.1

Схема организации файла с плотным индексом

Блок	Ключи	Ссылки на № записи	Свободная зона	Области
Блок 1	01-10/01	3		Индексная область
	02-20/02	4		
	03-20/00	5		
Блок 2	06-40/00	7		
	07-50/01	8		
	08-30/01	9		
Блок 3	10-44/01	1		
	11-44/02	2		
	09-35/01	6		
Блок 4	17-20/03	10		
	18-40/02	11		
	20-35/02	12		

Номер записи	Ключ	Содержание	Области
1	10-44/01	Математика	Основная область
2	11-44/02	Физика	
3	01-10/01	Информатика	
4	02-20/02	Теория информации	
5	03-20/00	Базы данных	
6	09-35/01	Интерфейсы АСОиУ	
7	06-40/00	Защита информации	
8	07-50/01	АСТПП и САПР	
9	08-30/01	Языки программирования	
10	17-20/03	Операционные системы	
11	18-40/02	Цифровые сети интегрального обслуживания	
12	20-35/02	Технологии программирования	

Рассмотрим, как осуществляются операции добавления и удаления новых записей в файлах с плотным индексом. В табл. 3.1 представлена схема организации такого файла на дисковом пространстве (фоном выделены свободные зоны).

Из табл. 3.1 видно, что файл организован в виде двух областей — основной и индексной. В основной области хранятся значения ключевых полей, номера и содержание записей. В индексной области хранятся значения ключевых полей и ссылки на номер записи в основной области.

При операции добавления осуществляется запись данных в конец основной области. При этом в индексную область необходимо добавить значения соответствующего ключевого поля и ссылку на номер записи, причем добавить информацию необходимо таким образом, чтобы не нарушить порядок записей. Поэтому вся индексная область файла разбивается на блоки и при начальном заполнении в каждом блоке создается свободная зона (пространство). В соответствующее свободное пространство соответствующего блока и заносится информация о добавленной записи.

Такой прием организации индексной области аналогичен системам многоуровневой классификации (буквенно-цифрового кодирования) изделий в системах ЕСКД, при которых на каждом уровне предусматривается заведомо большее число цифровых разрядов для описания характеристик изделий. Это позволяет без нарушения системы вводить новые типы изделий и присваивать им соответствующие буквенно-цифровые коды.

Именно поэтому при проектировании физической модели хранения данных необходимо как можно точнее определить объемы хранимой информации, спрогнозировать ее рост и соответственно предусмотреть соответствующее расширение области хранения.

При организации хранения данных в виде файлов с плотным индексом число обращений к диску при добавлении новой записи определится по формуле

$$T_д = \log_2 N_{инд.\ обл} + 1 + 1 + 1.$$

Смысл формулы заключается в следующем: число обращений определяется числом обращений к индексной области плюс одно обращение к основному блоку, плюс одно обращение для изменения индексного блока и плюс одно обращение для занесения записи в основную область.

Таким образом, в файлах с плотным индексом при обработке одной записи требуется дополнительно два обращения к дисковому пространству компьютера.

Следовательно, способы организации файлов баз данных и соответствующие им физические модели должны быть направлены на сокращение времени обращения к дисковому пространству при ее поиске и сокращению времени на добавление и корректировку

содержания баз данных. На это и направлен метод организации файлов с неплотным индексом.

Файлы с неплотным индексом, или индексно-последовательные файлы. Для уменьшения числа обращений к дисковому пространству в таких файлах применяют принцип внутреннего упорядочения организации записей. Отличие структурной организации таких файлов заключается в следующем. Индексная область не содержит ни блоков, ни свободного пространства. Зато основная область, в которой хранятся записи, распределена на блоки, имеющие свободное пространство. Все записи в таких файлах упорядочиваются по значению первичного ключа. Индексы, в которых исходные записи упорядочены по значениям первичного ключа, называются *кластеризованными*.

Структура записей данных в таких файлах имеет вид, представленный на рис. 3.4.

При такой организации файловой структуры процессы добавления новых записей отличаются от аналогичных действий в файлах с плотным индексом. Каждая новая запись заносится в соответственный блок на место, определенное значением ключевого поля. В этом случае выполняется следующая последовательность действий:

- определяется номер блока основной области, в который необходимо поместить новую запись;
- найденный блок считывается в оперативную память;
- в оперативной памяти производится корректировка блока;
- откорректированный блок записывается на диск на прежнее место.

В этом случае число обращений к диску при внесении новой записи равно числу обращений к диску при поиске блока плюс одно обращение, которое необходимо выполнить при записи откорректированного блока на прежнее место. В данном случае не принимается во внимание время записи блока в оперативную память, которое несопоставимо с временем обращения к диску.

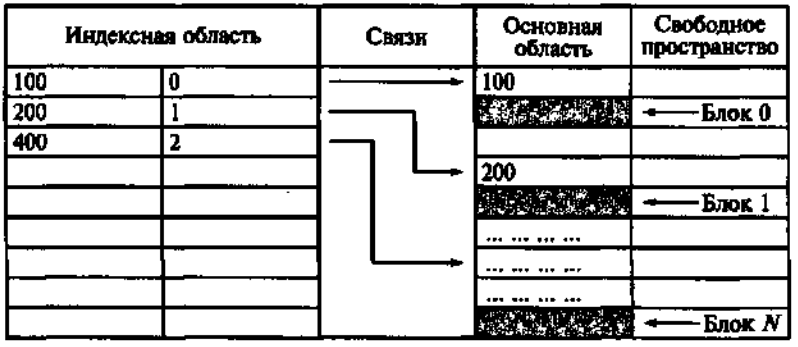


Рис. 3.4. Структура записей данных в файлах с неплотным индексом

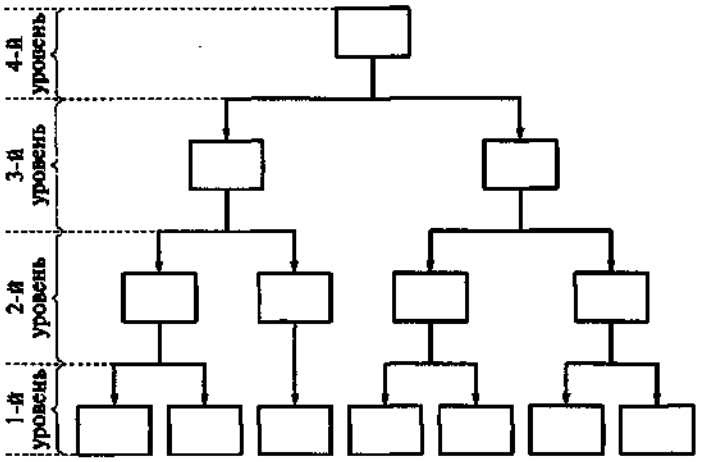


Рис. 3.5. Пример организации файловой структуры в виде *B*-дерева

Следовательно, число обращений к дисковому пространству при такой организации файловой структуры будет на единицу меньше, чем у фалов с плотным индексом для каждой записи, что при значительном числе записей не только существенно сокращает время обработки данных, но и повышает надежность работы дисковых устройств.

Организация индексов в виде *B*-дерева — многоуровневой иерархической структуры. Данное направление совершенствования организации файловой структуры связано с преобразованием индексной области файлов с неплотным индексом, который изначально предполагает описание этой области как одного упорядоченного списка, в вид иерархического симметрического поискового дерева. В таких деревьях число узлов на каждом уровне одинаково. Теоретические основы организации машинной памяти при построении таких иерархических систем были изложены в 1967 г. автором языка ассоциативного программирования АЛГЭМ, преподавателем Московского энергетического института А. И. Китовым.

Однако в современной литературе по теории баз данных иерархическую поисковую структуру принято называть *B*-деревом (читается: «Б-деревом») (от англ. *B-tree* — сбалансированное дерево).

На рис. 3.5 показан пример организации файловой структуры в виде *B*-дерева.

3.5. Способы организации памяти для хранения данных

В основе реализации организации памяти современных компьютеров лежат два принципа: принцип локальности обращений и соотношение стоимость/производительность. Принцип локально-

сти обращений говорит о том, что большинство программ не выполняют обращений ко всем своим командам и данным равновероятно, а оказывают предпочтение некоторой части своего адресного пространства. Рассмотрим следующие аспекты организации памяти для хранения данных:

- иерархическая организации памяти;
- организация кэш-памяти;
- организация основной памяти;
- виртуальная память — как средство организации защиты данных.

Иерархическая организация памяти. Иерархическая организация памяти современных компьютеров строится на нескольких уровнях, причем более высокий уровень меньше по объему, быстрее и имеет бóльшую стоимость в пересчете на байт, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на более низком уровне, и все данные на этом более низком уровне могут быть найдены на следующем лежащем ниже уровне и так далее, пока мы не достигнем основания иерархии.

Иерархия памяти обычно состоит из многих уровней, но в каждый момент времени мы имеем дело только с двумя близлежащими уровнями. Минимальная единица информации, которая может либо присутствовать, либо отсутствовать в двухуровневой иерархии, называется *блоком*. Размер блока может быть либо фиксированным, либо переменным. Если этот размер зафиксирован, то объем памяти является кратным размеру блока.

Успешное или неуспешное обращение к более высокому уровню называются соответственно попаданием (hit) или промахом (miss). Попаданне есть обращение к объекту в памяти, который найден на более высоком уровне, в то время как промах означает, что он не найден на этом уровне. Доля попаданий (hit rate), или коэффициент попаданий (hit ratio), есть доля обращений, найденных на более высоком уровне. Иногда она выражается в процентах. Доля промахов (miss rate) есть доля обращений, которые не найдены на более высоком уровне.

Поскольку повышение производительности обработки информации является главной причиной появления иерархии памяти, частота попаданий и промахов является важной характеристикой. Время обращения при попадании (hit time) есть время обращения к более высокому уровню иерархии, которое включает в себя, в частности, и время, необходимое для определения того, является ли обращение попаданием или промахом. Потери на промах (miss penalty) есть время для замещения блока в более высоком уровне на блок из более низкого уровня плюс время для пересылки этого блока в требуемое устройство (обычно в процессор). Потери на промах далее включают в себя две компоненты: время

доступа (access time) — время обращения к первому слову блока при промахе, и время пересылки (transfer time) — дополнительное время для пересылки оставшихся слов блока. Время доступа связано с задержкой памяти более низкого уровня, в то время как время пересылки связано с полосой пропускания канала между устройствами памяти двух смежных уровней.

Чтобы описать некоторый уровень иерархии памяти, надо ответить на четыре вопроса.

1. Где может размещаться блок на верхнем уровне иерархии (размещение блока)?

2. Как найти блок, когда он находится на верхнем уровне (идентификация блока)?

3. Какой блок должен быть замещен в случае промаха (замещение блоков)?

4. Что происходит во время записи (стратегия записи)?

Ответы на эти вопросы связаны с организацией кэш-памяти.

Организация кэш-памяти. Концепция кэш-памяти возникла раньше, чем архитектура IBM/360. Сегодня кэш-память имеется практически в любом классе компьютеров, а в некоторых компьютерах — во множественном числе.

Все термины, приведенные выше, могут быть использованы и для кэш-памяти, хотя слово «строка» («line») часто употребляется вместо слова «блок» («block»).

Типичный набор параметров описания кэш-памяти

Размер блока (строки)	4...128 байт
Время попадания (hit time)	1—4 такта синхронизации (обычно 1 такт)
Потери при промахе (miss penalty)	8—32 такта синхронизации
Время доступа (access time)	6—18 тактов синхронизации
Время пересылки (transfer time)	2—22 такта синхронизации
Доля промахов (miss rate)	1...20 %
Размер кэш-памяти	4 Кбайт... 16 Мбайт

Рассмотрим организацию кэш-памяти более детально, отвечая на поставленные выше вопросы об иерархии памяти.

1. *Где может размещаться блок в кэш-памяти?* Принципы размещения блоков в кэш-памяти определяют три основных типа их организации:

• если каждый блок основной памяти имеет только одно фиксированное место, на котором он может появиться в кэш-памяти, то такая кэш-память называется кэшем с *прямым отображением* (direct mapped). Это наиболее простая организация кэш-па-

мяти, при которой для отображения адресов блоков основной памяти на адреса кэш-памяти просто используются младшие разряды адреса блока. Таким образом, все блоки основной памяти, имеющие одинаковые младшие разряды в своем адресе, попадают в один блок кэш-памяти, т.е.

(адрес блока кэш-памяти) = (адрес блока основной памяти) × × mod (число блоков в кэш-памяти);

• если некоторый блок основной памяти может располагаться на любом месте кэш-памяти, то кэш называется *полностью ассоциативным* (fully associative);

• если некоторый блок основной памяти может располагаться на ограниченном множестве мест в кэш-памяти, то кэш называется *множественно-ассоциативным* (set associative). Обычно множество представляет собой группу из двух или большего числа блоков в кэше. Если множество состоит из *n* блоков, то такое размещение называется множественно-ассоциативным с *n* каналами (*n*-way set associative). Для размещения блока прежде всего необходимо определить множество. Множество определяется младшими разрядами адреса блока памяти (индексом):

(адрес множества кэш-памяти) = (адрес блока основной памяти) · mod (число множеств в кэш-памяти).

Блок может размещаться на любом месте данного множества.

Диапазон возможных организаций кэш-памяти очень широк: кэш-память с прямым отображением есть просто одноканальная множественно-ассоциативная кэш-память, а полностью ассоциативная кэш-память с *m* блоками может быть названа *m*-канальной множественно-ассоциативной. В современных процессорах, как правило, используется либо кэш-память с прямым отображением, либо двухканальная (четырехканальная) множественно-ассоциативная кэш-память.

2. *Как найти блок, находящийся в кэш-памяти?* У каждого блока в кэш-памяти имеется адресный тег, указывающий, какой блок в основной памяти данный блок кэш-памяти представляет. Эти теги обычно одновременно сравниваются с выработанным процессором адресом блока памяти.

Кроме того, необходим способ определения того, что блок кэш-памяти содержит достоверную или пригодную для использования информацию. Наиболее общим способом решения этой проблемы является добавление к тегу так называемого бита достоверности (valid bit).

Адресация множественно-ассоциативной кэш-памяти осуществляется путем деления адреса, поступающего из процессора, на три части: поле смещения, которое используется для выбора байта внутри блока кэш-памяти; поле индекса, которое определяет номер

множества; поле тега, которое используется для сравнения. Если общий размер кэш-памяти зафиксировать, то увеличение степени ассоциативности приведет к увеличению числа блоков в множестве; при этом уменьшится размер индекса и увеличится размер тега;

3. *Какой блок кэш-памяти должен быть замещен при промахе?* При возникновении промаха контроллер кэш-памяти должен выбрать подлежащий замещению блок. Польза от использования организации с прямым отображением заключается в том, что аппаратные решения здесь наиболее простые. Выбирать просто нечего: на попадание проверяется только один блок и только этот блок может быть замещен. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Как правило, для замещения блоков применяются две основные стратегии: случайная и Least-Recently Used (LRU).

В первом случае, чтобы иметь равномерное распределение, блоки-кандидаты выбираются случайно. В некоторых системах, чтобы получить воспроизводимое поведение, которое особенно полезно во время отладки аппаратуры, используют псевдослучайный алгоритм замещения.

Во втором случае, чтобы уменьшить вероятность выбрасывания информации, которая скоро может потребоваться, все обращения к блокам фиксируются. Заменяется тот блок, который не использовался дольше всех.

Достоинство случайного способа заключается в том, что его проще реализовать в аппаратуре. Когда число блоков для поддержания трассы увеличивается, алгоритм LRU становится все более дорогим и часто только приближенным. В табл. 3.2 показаны различия в долях промахов при использовании алгоритма замещения LRU и случайного алгоритма.

4. *Что происходит во время записи?* При обращениях к кэш-памяти на реальных программах преобладают обращения по чте-

Таблица 3.2

Доля промахов при использовании алгоритма замещения LRU и случайного алгоритма (Random)

Размер кэш-памяти, Кбайт	Ассоциативность, %					
	2-канальная		4-канальная		8-канальная	
	LRU	Random	LRU	Random	LRU	Random
16	5,18	5,69	4,67	5,29	4,39	4,96
64	1,88	2,01	1,54	1,66	1,39	1,53
256	1,15	1,17	1,13	1,13	1,12	1,12

нию. Все обращения за командами являются обращениями по чтению и большинство команд не пишут в память. Обычно операции записи составляют менее 10 % от общего трафика памяти. Желание сделать общий случай более быстрым означает оптимизацию кэш-памяти для выполнения операций чтения, однако при реализации высокопроизводительной обработки данных нельзя пренебрегать и скоростью операций записи.

Общий случай является и более простым. Блок из кэш-памяти может быть прочитан в то же самое время, в которое читается и сравнивается его тег. Таким образом, чтение блока начинается сразу, как только становится доступным адрес блока. Если чтение происходит с попаданием, то блок немедленно направляется в процессор. Если же происходит промах, то от заранее считанного блока нет никакой пользы, правда, нет и никакого вреда.

Однако при выполнении операции записи ситуация коренным образом меняется. Именно процессор определяет размер записи (обычно от 1 до 8 байт), и только эта часть блока может быть изменена. В общем случае это подразумевает выполнение над блоком последовательности операций чтение — модификация — запись (чтение оригинала блока, модификация его части и запись нового значения блока). Более того, модификация блока не может начаться, пока проверяется тег, чтобы убедиться в том, что обращение является попаданием. Поскольку проверка тегов не может выполняться параллельно с другой работой, то операции записи отнимают больше времени, чем операции чтения.

Очень часто организация кэш-памяти в разных машинах отличается именно стратегией выполнения записи. Когда выполняется запись в кэш-память, имеются две базовые возможности:

- сквозная запись (write through, store through) — информация записывается в два места (в блок кэш-памяти и блок более низкого уровня памяти);
- запись с обратным копированием (write back, copy back, store in) — информация записывается только в блок кэш-памяти. Модифицированный блок кэш-памяти записывается в основную память только тогда, когда он замещается. Для сокращения частоты копирования блоков при замещении обычно с каждым блоком кэш-памяти связывается так называемый бит модификации (dirty bit). Этот бит состояния показывает, был ли модифицирован блок, находящийся в кэш-памяти. Если он не модифицировался, то обратное копирование отменяется, поскольку более низкий уровень содержит ту же самую информацию, что и кэш-память.

Оба подхода к организации записи имеют свои преимущества и недостатки. При записи с обратным копированием операции записи выполняются со скоростью кэш-памяти и несколько записей в один и тот же блок требуют только одной записи в память более низкого уровня. Поскольку в этом случае обращения к основной

памяти происходят реже, то требуется и меньшая полоса пропускания памяти, что очень привлекательно для мультипроцессорных систем. При сквозной записи промахи по чтению не влияют на записи в более высокий уровень. Кроме того, сквозная запись проще для реализации, чем запись с обратным копированием. Еще одним преимуществом сквозной записи является то, что основная память имеет наиболее свежую копию данных. Это важно в мультипроцессорных системах, а также для организации ввода (вывода).

Когда процессор ожидает завершения записи при выполнении сквозной записи, то говорят, что он приостанавливается для записи (write stall). Общий прием минимизации остановов по записи связан с использованием буфера записи (write buffer), который позволяет процессору продолжить выполнение команд во время обновления содержимого памяти. Следует отметить, что остановы по записи могут возникать и при наличии буфера записи.

При промахе во время записи имеются две дополнительные возможности: разместить запись в кэш-памяти и не размещать запись в кэш-памяти.

Разместить запись в кэш-памяти (write allocate) (называется также выборкой при записи (fetch on write)) означает, что блок загружается в кэш-память, вслед за чем выполняются действия, аналогичные выполняющимся при выполнении записи с попаданием. Это похоже на промах при чтении.

Не размещать запись в кэш-памяти (называется также записью в окружение (write around)), означает, что блок модифицируется на более низком уровне и не загружается в кэш-память.

Обычно в кэш-памяти, реализующей запись с обратным копированием, используется размещение записи в кэш-памяти (в надежде, что последующая запись в этот блок будет перехвачена), а в кэш-памяти со сквозной записью размещение записи в кэш-памяти часто не используется (поскольку последующая запись в этот блок все равно пойдет в память).

Рассмотрим способы увеличения производительности кэш-памяти. Формула для среднего времени доступа к памяти в системах с кэш-памятью выглядит следующим образом:

$$\text{Среднее время доступа} = \text{Время обращения при попадании} + \text{Доля промахов} \times \text{Потери при промахе}.$$

Эта формула наглядно показывает пути оптимизации работы кэш-памяти: сокращение доли промахов, сокращение потерь при промахе, а также сокращение времени обращения к кэш-памяти при попадании.

В табл. 3.3 кратко представлены различные методы, которые используются в настоящее время для увеличения производительности кэш-памяти. Использование тех или иных методов опреде-

Таблица 3.3

Методы увеличения производительности кэш-памяти

Метод	Доля промахов	Потери при промахе	Время обращения при попадании (примечания)
Увеличение размера блока	+	–	0
Повышение степени ассоциативности	+		–1
Кэш-память с вспомогательным кэшем	+		2
Псевдоассоциативные кэши	+		2
Аппаратная предварительная выборка команд и данных	+		2 (предварительная выборка данных затруднена)
Предварительная выборка под управлением компилятора	+		3 (требует также неблокируемой кэш-памяти)
Специальные методы для уменьшения промахов	+		0 (вопрос программного обеспечения)
Установка приоритетов промахов по чтению над записями		+	1 (просто для однопроцессорных систем)
Использование подблоков		+	+1 (сквозная запись + + подблок на одно слово помогают записям)
Пересылка требуемого слова первым		+	2
Неблокируемые кэши		+	3
Кэши второго уровня		+	2 (достаточно дорогое оборудование)
Простые кэши малого размера	–		0
Обход преобразования адресов во время индексации кэш-памяти			+2
Конвейеризация операций записи для быстрого попадания при записи			+1

ляется прежде всего целью разработки; при этом конструкторы современных компьютеров заботятся о том, чтобы система оказалась сбалансированной по всем параметрам.

Организация основной памяти. Основная память в современных компьютерах представляет собой следующий уровень иерархии памяти. Основная память удовлетворяет запросы кэш-памяти и служит в качестве интерфейса ввода (вывода), поскольку является местом назначения для ввода и источником для вывода. Для оценки производительности основной памяти используются два основных параметра: задержка и полоса пропускания. Традиционно задержка основной памяти имеет отношение к кэш-памяти, а полоса пропускания, или пропускная способность, относится к вводу (выводу). В связи с ростом популярности кэш-памяти второго уровня и увеличением размеров блоков у такой кэш-памяти полоса пропускания основной памяти становится важной также и для кэш-памяти.

Задержка памяти традиционно оценивается двумя параметрами: временем доступа (access time) и длительностью цикла памяти (cycle time). Время доступа представляет собой промежуток времени между выдачей запроса на чтение и моментом поступления запрошенного слова из памяти. Длительность цикла памяти определяется минимальным временем между двумя последовательными обращениями к памяти.

Основная память современных компьютеров реализуется на микросхемах статических и динамических запоминающих устройств с произвольной выборкой (ЗУПВ). Микросхемы статических ЗУПВ (СЗУПВ) имеют меньшее время доступа и не требуют циклов регенерации. Микросхемы динамических ЗУПВ (ДЗУПВ) характеризуются большей емкостью и меньшей стоимостью, но требуют схем регенерации и имеют значительно большее время доступа.

В процессе развития ДЗУПВ с ростом их емкости основным вопросом стоимости таких микросхем был вопрос о числе адресных линий и стоимости соответствующего корпуса. Было принято решение о необходимости мультиплексирования адресных линий, позволившее сократить наполовину число контактов корпуса, необходимых для передачи адреса. Поэтому обращение к ДЗУПВ обычно происходит в два этапа. Первый этап начинается с выдачи сигнала RAS (row-access strobe — строб адреса строки), который фиксирует в микросхеме поступивший адрес строки. Второй этап включает в себя переключение адреса для указания адреса столбца и подачу сигнала CAS (column-access stobe — строб адреса столбца), который фиксирует этот адрес и разрешает работу выходных буферов микросхемы. Названия этих сигналов связаны с внутренней организацией микросхемы, представляющей собой, как правило, прямоугольную матрицу, к элементам ко-

торой можно адресоваться с помощью указания адреса строки и адреса столбца.

Дополнительным требованием организации ДЗУПВ является необходимость периодической регенерации ее состояния. При этом все биты в строке могут регенерироваться одновременно, например путем чтения этой строки. Поэтому ко всем строкам всех микросхем ДЗУПВ основной памяти компьютера должны производиться периодические обращения в пределах определенного временного интервала — порядка 8 мс.

Это требование означает, что система основной памяти компьютера оказывается иногда недоступной процессору, так как она вынуждена рассылать сигналы регенерации каждой микросхеме. Разработчики ДЗУПВ стараются поддерживать время, затрачиваемое на регенерацию, на уровне менее 5 % от общего времени. Обычно контроллеры памяти включают в свой состав аппаратуру для периодической регенерации ДЗУПВ.

В отличие от динамических статические ЗУПВ не требуют регенерации, и время доступа к ним совпадает с длительностью цикла. Для микросхем, использующих примерно одну и ту же технологию, емкость ДЗУПВ в 4—8 раз превышает емкость СЗУПВ, но последние имеют в 8 — 16 раз меньшую длительность цикла и большую стоимость. По этим причинам в основной памяти практически любого компьютера, проданного после 1975 г., использовались полупроводниковые микросхемы ДЗУПВ (для построения кэш-памяти при этом применялись СЗУПВ). Но были и исключения. Например, в оперативной памяти суперкомпьютеров компании Cray Research использовались микросхемы СЗУПВ.

Для обеспечения сбалансированности системы с ростом скорости процессоров должна линейно расти и емкость основной памяти. В последние годы емкость микросхем динамической памяти учетверялась каждые три года, увеличиваясь примерно на 60 % в год. К сожалению, скорость этих схем за этот же период росла гораздо меньшими темпами (примерно на 7 % в год). В то же время производительность процессоров, начиная с 1987 г., увеличивалась практически на 50 % в год.

Согласование производительности современных процессоров со скоростью основной памяти вычислительных систем остается одной из важнейших проблем. Приведенные в подразд. 3.4 методы повышения производительности обработки информации за счет увеличения размеров кэш-памяти и введения многоуровневой организации кэш-памяти могут оказаться не достаточно эффективными с точки зрения стоимости систем. Поэтому важным направлением современных разработок являются методы повышения полосы пропускания, или пропускной способности памяти, за счет ее организации, включая специальные методы организации ДЗУПВ.

Для организации кэш-памяти в большей степени важно уменьшение задержки памяти, чем увеличение полосы пропускания. Однако при увеличении полосы пропускания памяти возможно увеличение размера блоков кэш-памяти без заметного увеличения потерь при промахах.

Основными методами увеличения полосы пропускания памяти являются: увеличение разрядности, или «ширины» памяти; использование расслоения памяти; использование независимых банков памяти; обеспечение режима бесконфликтного обращения к банкам памяти; использование специальных режимов работы динамических микросхем памяти.

Увеличение разрядности основной памяти. Кэш-память первого уровня во многих случаях имеет физическую ширину шин данных, соответствующую числу разрядов в слове, поскольку большинство компьютеров выполняют обращения именно к этой единице информации. В системах без кэш-памяти второго уровня ширина шин данных основной памяти часто соответствует ширине шин данных кэш-памяти. Удвоение или учетверение ширины шин кэш-памяти и основной памяти удваивает или учетверяет соответственно полосу пропускания системы памяти.

Реализация более широких шин вызывает необходимость мультиплексирования данных между кэш-памятью и процессором, поскольку основной единицей обработки данных в процессоре все еще остается слово. Эти мультиплексоры оказываются на критическом пути поступления информации в процессор. Кэш-память второго уровня несколько смягчает эту проблему, так как в этом случае мультиплексоры могут располагаться между двумя уровнями кэш-памяти, т.е. вносимая ими задержка не столь критична. Другая проблема, связанная с увеличением разрядности памяти, определяется необходимостью определения минимального объема (инкремента) для поэтапного расширения памяти, которое часто выполняется самими пользователями на месте эксплуатации системы. Удвоение или учетверение ширины памяти приводит к удвоению или учетверению этого минимального инкремента. Имеются также проблемы с организацией коррекции ошибок в системах с широкой памятью.

Память с расслоением. Наличие в системе множества микросхем памяти позволяет использовать потенциальный параллелизм, заложенный в такой организации. Для этого микросхемы памяти часто объединяются в банки, или модули, содержащие фиксированное число слов, причем только к одному из этих слов банка возможно обращение в каждый момент времени. Как уже отмечалось, в реальных системах имеющаяся скорость доступа к таким банкам памяти редко оказывается достаточной. Следовательно, чтобы получить большую скорость доступа, нужно осуществлять одновременный доступ ко многим банкам памяти. Одна из общих

методик, используемых для этого, называется расслоением памяти. При расслоении банки памяти обычно упорядочиваются так, чтобы N последовательных адресов памяти i, $i + 1$, $i + 2$, ..., $i + (N - 1)$ приходились на N различных банков. В i-м банке памяти находятся только слова, адреса которых имеют вид

$$kN + i,$$

где $k = M - 1$ (M — число слов в одном банке).

Можно достичь в N раз большей скорости доступа к памяти в целом, чем у отдельного ее банка, если обеспечить при каждом доступе обращение к данным в каждом из банков. Имеются разные способы реализации таких расслоенных структур. Большинство из них напоминают конвейеры, обеспечивающие рассылку адресов в различные банки и мультиплексируюшие поступающие из банков данные. Таким образом, степень, или коэффициент, расслоения определяет распределение адресов по банкам памяти. Такие системы оптимизируют обращения по последовательным адресам памяти, что является характерным при подкачке информации в кэш-память при чтении, а также при записи в случае использования кэш-памятью механизмов обратного копирования. Однако если требуется доступ к непоследовательно расположенным словам памяти, производительность расслоенной памяти может значительно снижаться.

Обобщением идеи расслоения памяти является возможность реализации нескольких независимых обращений, когда несколько контроллеров памяти позволяют банкам памяти (или группам расслоенных банков памяти) работать независимо.

Если система памяти разработана для поддержки множества независимых запросов (как это имеет место при работе с кэш-памятью, при реализации многопроцессорной и векторной обработки), эффективность системы будет в значительной степени зависеть от частоты поступления независимых запросов к разным банкам. Обращения по последовательным адресам, или в более общем случае обращения по адресам, отличающимся на нечетное число, хорошо обрабатываются традиционными схемами расслоенной памяти. Проблемы появляются, если разница в адресах последовательных обращений четная. Одно из решений, используемых в больших компьютерах, заключается в том, чтобы статистически уменьшить вероятность подобных обращений путем значительного увеличения числа банков памяти. Например, в суперкомпьютере NEC SX/3 используются 128 банков памяти.

Подобные проблемы могут быть решены как программными, так и аппаратными средствами.

Использование специфических свойств ДЗУПВ. Обращение к ДЗУПВ состоит из двух этапов: обращение к строке и обращение к

столбцу. При этом внутри микросхемы осуществляется буферизация битов строки, прежде чем происходит обращение к столбцу. Размер строки обычно является корнем квадратным от емкости кристалла памяти: 1024 бита для 1Мбит, 2048 бит для 4 Мбит и т.д. С целью увеличения производительности все современные микросхемы памяти обеспечивают возможность подачи сигналов синхронизации, которые позволяют выполнять последовательные обращения к буферу без дополнительного времени обращения к строке. Имеются три способа подобной оптимизации:

- блочный режим;
- страничный режим;
- режим статического столбца.

Блочный режим (nibble mode) может обеспечить выдачу четырех последовательных ячеек для каждого сигнала RAS.

При страничном режиме (page mode) буфер работает как статическое ЗУПВ; при изменении адреса столбца возможен доступ к произвольным битам в буфере до тех пор, пока не поступит новое обращение к строке или не наступит время регенерации.

Режим статического столбца (static column) очень похож на страничный режим, за исключением того, что не обязательно переключать строб адреса столбца каждый раз для изменения адреса столбца.

Начиная с микросхем ДЗУПВ емкостью 1 Мбит, большинство ДЗУПВ допускают любой из этих режимов, причем выбор режима осуществляется на стадии установки кристалла в корпус путем выбора соответствующих соединений. Эти операции изменили определение длительности цикла памяти для ДЗУПВ. Преимуществом такой оптимизации является то, что она основана на внутренних схемах ДЗУПВ и незначительно увеличивает стоимость системы, позволяя практически учетверить пропускную способность памяти. Например, nibble mode был разработан для поддержки режимов, аналогичных расслоению памяти. Кристалл за один раз читает значения четырех бит и подает их наружу в течение четырех оптимизированных циклов. Если время пересылки по шине не превышает время оптимизированного цикла, единственное усложнение для организации памяти с четырехкратным расслоением заключается в несколько усложненной схеме управления синхросигналами. Страничный режим и режим статического столбца также могут использоваться, обеспечивая даже большую степень расслоения при несколько более сложном управлении. Одной из тенденций в разработке ДЗУПВ является наличие в них буферов с тремя состояниями. Это предполагает, что для реализации традиционного расслоения с большим числом кристаллов памяти в системе должны быть предусмотрены буферные микросхемы для каждого банка памяти.

Новые поколения ДЗУПВ разработаны с учетом возможности дальнейшей оптимизации интерфейса между ДЗУПВ и процессором. В качестве примера можно привести изделия компании RAMBUS. Эта компания берет стандартную начинку ДЗУПВ и обеспечивает новый интерфейс, делающий работу отдельной микросхемы более похожей на работу системы памяти, а не на работу отдельного ее компонента. RAMBUS отбросила сигналы RAS/CAS, заменив их шиной, которая допускает выполнение других обращений в интервале между посылкой адреса и приходом данных. Такого рода шины называются шинами с пакетным переключением (packet-switched bus), или шинами с расщепленными транзакциями (split-trancaction bus). Такая шина позволяет работать кристаллу как отдельному банку памяти. Кристалл может вернуть переменное число данных на один запрос и даже самостоятельно выполняет регенерацию. RAMBUS предлагает байтовый интерфейс и сигнал синхронизации, так что микросхема может тесно синхронизироваться с тактовой частотой процессора. После того как адресный конвейер наполнен, отдельный кристалл может выдавать по байту каждые 2 нс.

Большинство систем основной памяти используют методы, подобные страничному режиму ДЗУПВ, для уменьшения различий в производительности процессоров и микросхем памяти.

Виртуальная память — как средство организации защиты данных. Общепринятая в настоящее время концепция виртуальной памяти появилась достаточно давно. Она позволила решить целый ряд актуальных вопросов организации вычислений, к числу которых относится обеспечение надежного функционирования мультипрограммных систем.

В любой момент времени компьютер выполняет множество процессов или задач, каждая из которых располагает своим адресным пространством. Было бы слишком накладно отдавать всю физическую память какой-то одной задаче, тем более, что многие задачи реально используют только небольшую часть своего адресного пространства. Поэтому необходим механизм разделения небольшой физической памяти между различными задачами. Виртуальная память является одним из способов реализации такой возможности. Она делит физическую память на блоки и распределяет их между различными задачами. При этом она предусматривает также некоторую схему защиты, которая ограничивает задачу теми блоками, которые ей принадлежат. Большинство типов виртуальной памяти сокращают также время начального запуска программы на процессоре, поскольку не весь программный код и данные требуются ей в физической памяти, чтобы начать выполнение.

Другой вопрос, тесно связанный с реализацией концепции виртуальной памяти, касается организации вычислений задач очень

большого объема. Если программа становилась слишком большой для физической памяти, то часть ее необходимо было хранить во внешней памяти (на диске) и задача приспособления ее для решения на компьютере ложилась на программиста. Программисты делили программы на части и затем определяли те из них, которые можно было бы выполнять независимо, организуя оверлейные структуры, которые загружались в основную память и выгружались из нее под управлением программы пользователя. Программист должен был следить за тем, чтобы программа не обращалась вне отведенного ей пространства физической памяти. Виртуальная память освободила программистов от этого бремени. Она автоматически управляет двумя уровнями иерархии памяти: основной и внешней (дисковой) памятью.

Кроме того, виртуальная память упрощает загрузку программ, обеспечивая механизм автоматического перемещения программ, позволяющий выполнять одну и ту же программу в произвольном месте физической памяти.

Системы виртуальной памяти можно разделить на два типа: системы с фиксированным размером блоков, называемых страницами, и системы с переменным размером блоков, называемых сегментами. Рассмотрим оба типа организации виртуальной памяти.

Страничная организация памяти. В системах со страничной организацией основная и внешняя память (главным образом дисковое пространство) делятся на блоки, или страницы, фиксированной длины. Каждому пользователю предоставляется некоторая часть адресного пространства, которая может превышать основную память компьютера и которая ограничена только возможностями адресации, заложенными в системе команд. Эта часть адресного пространства называется виртуальной памятью пользователя. Каждое слово в виртуальной памяти пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер страницы, а младшие — как номер слова (или байта) внутри страницы.

Управление различными уровнями памяти осуществляется программами ядра операционной системы, которые следят за распределением страниц и оптимизируют обмены между этими уровнями. При страничной организации памяти смежные виртуальные страницы не обязательно должны размещаться на смежных страницах основной физической памяти. Для указания соответствия между виртуальными страницами и страницами основной памяти операционная система должна сформировать таблицу страниц для каждой программы и разместить ее в основной памяти машины. При этом каждой странице программы, независимо от того, находится ли она в основной памяти или нет, ставится в соответствие некоторый элемент таблицы страниц. Каждый эле-

мент таблицы страниц содержит номер физической страницы основной памяти и специальный индикатор. Единичное состояние этого индикатора свидетельствует о наличии этой страницы в основной памяти. Нулевое состояние индикатора означает отсутствие страницы в оперативной памяти.

Для увеличения эффективности такого типа схем в процессорах используется специальная полностью ассоциативная кэш-память, которая также называется буфером преобразования адресов (TLB — tranclation-lookaside buffer). Хотя наличие TLB не меняет принципа построения схемы страничной организации, с точки зрения защиты памяти необходимо предусмотреть возможность очистки его при переключении с одной программы на другую.

Поиск в таблицах страниц, расположенных в основной памяти, и загрузка TLB могут осуществляться либо программным способом, либо специальными аппаратными средствами. В последнем случае для того, чтобы предотвратить возможность обращения пользовательской программы к таблицам страниц, с которыми она не связана, предусмотрены специальные меры. С этой целью в процессоре предусматривается дополнительный регистр защиты, содержащий описатель (дескриптор) таблицы страниц или базово-граничную пару. База определяет адрес начала таблицы страниц в основной памяти, а граница — длину таблицы страниц соответствующей программы. Загрузка этого регистра защиты разрешена только в привилегированном режиме. Для каждой программы операционная система хранит дескриптор таблицы страниц и устанавливает его в регистр защиты процессора перед запуском соответствующей программы.

Отметим некоторые особенности, присущие простым схемам со страничной организацией памяти. Наиболее важной из них является то, что все программы, которые должны непосредственно связываться друг с другом без вмешательства операционной системы, должны использовать общее пространство виртуальных адресов. Это относится и к самой операционной системе, которая должна работать в режиме динамического распределения памяти. Поэтому в некоторых системах пространство виртуальных адресов пользователя укорачивается на размер общих процедур, к которым программы пользователей желают иметь доступ. Общим процедурам должен быть отведен определенный объем пространства виртуальных адресов всех пользователей, чтобы они имели постоянное место в таблицах страниц всех пользователей. В этом случае для обеспечения целостности, секретности и взаимной изоляции выполняющихся программ должны быть предусмотрены различные режимы доступа к страницам, которые реализуются с помощью специальных индикаторов доступа в элементах таблиц страниц.

Следствием такого использования является значительный рост таблиц страниц каждого пользователя. Одно из решений пробле-

мы сокращения длины таблиц основано на введении многоуровневой организации таблиц. Частным случаем многоуровневой организации таблиц является сегментация при страничной организации памяти. Необходимость увеличения адресного пространства пользователя объясняется желанием избежать необходимости перемещения частей программ и данных в пределах адресного пространства, которые обычно приводят к проблемам переименования и серьезным затруднениям в разделении общей информации между многими задачами.

Сегментация памяти. Другой подход к организации памяти опирается на тот факт, что программы обычно разделяются на отдельные области-сегменты. Каждый сегмент представляет собой отдельную логическую единицу информации, содержащую совокупность данных или программ и расположенную в адресном пространстве пользователя. Сегменты создаются пользователями, которые могут обращаться к ним по символическому имени. В каждом сегменте устанавливается своя собственная нумерация слов, начиная с нуля.

Обычно в подобных системах обмен информацией между пользователями строится на базе сегментов. Поэтому сегменты являются отдельными логическими единицами информации, которые необходимо защищать, и именно на этом уровне вводятся различные режимы доступа к сегментам. Можно выделить два основных типа сегментов: программные сегменты и сегменты данных (сегменты стека являются частным случаем сегментов данных). Поскольку общие программы должны обладать свойством повторной входимости, то из программных сегментов допускается только выборка команд и чтение констант. Запись в программные сегменты может рассматриваться как незаконная и запрещаться системой. Выборка команд из сегментов данных также может считаться незаконной, и любой сегмент данных может быть защищен от обращений по записи или по чтению.

Для реализации сегментации было предложено несколько схем, которые отличаются деталями реализации, но основаны на одних и тех же принципах.

В системах с сегментацией памяти каждое слово в адресном пространстве пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер сегмента, а младшие — как номер слова внутри сегмента. Наряду с сегментацией может также использоваться страничная организация памяти. В этом случае виртуальный адрес слова состоит из трех частей: старшие разряды адреса определяют номер сегмента, средние — номер страницы внутри сегмента, а младшие — номер слова внутри страницы.

Как и в случае страничной организации, необходимо обеспечить преобразование виртуального адреса в реальный физический

адрес основной памяти. С этой целью для каждого пользователя операционная система должна сформировать таблицу сегментов. Каждый элемент таблицы сегментов содержит описатель (дескриптор) сегмента (поля базы, границы и индикаторов режима доступа). При отсутствии страничной организации поле базы определяет адрес начала сегмента в основной памяти, а граница — длину сегмента. При наличии страничной организации поле базы определяет адрес начала таблицы страниц данного сегмента, а граница — число страниц в сегменте. Поле индикаторов режима доступа представляет собой некоторую комбинацию признаков блокировки чтения, записи и выполнения.

Таблицы сегментов различных пользователей операционная система хранит в основной памяти. Для определения расположения таблицы сегментов выполняющейся программы используется специальный регистр защиты, который загружается операционной системой перед началом ее выполнения. Этот регистр содержит дескриптор таблицы сегментов (базу и границу), причем база содержит адрес начала таблицы сегментов выполняющейся программы, а граница — длину этой таблицы сегментов. Разряды номера сегмента виртуального адреса используются в качестве индекса для поиска в таблице сегментов. Таким образом, наличие базовограничных пар в дескрипторе таблицы сегментов и элементах таблицы сегментов предотвращает возможность обращения программы пользователя к таблицам сегментов и страниц, с которыми она не связана. Наличие в элементах таблицы сегментов индикаторов режима доступа позволяет осуществлять необходимый режим доступа к сегменту со стороны данной программы. Для повышения эффективности схемы используется ассоциативная кэш-память.

В описанной схеме сегментации таблица сегментов с индикаторами доступа предоставляет всем программам, являющимся частями некоторой задачи, одинаковые возможности доступа, т.е. она определяет единственную область (домен) защиты. Однако для создания защищенных подсистем в рамках одной задачи, чтобы изменять возможности доступа, когда точка выполнения переходит через различные программы, управляющие ее решением, необходимо связать с каждой задачей множество доменов защиты. Реализация защищенных подсистем требует разработки некоторых специальных аппаратных средств.

Контрольные вопросы

1. Какие уровни описания информационной системы (типы моделей) предусматривает информационная модель данных?
2. Дайте характеристики каждому типу информационной модели данных.
3. Дайте краткую характеристику файловым структурам организации баз данных.

4. Укажите состав информации, характеризующий каждый файл, который хранится в файловых структурах организации баз данных.

5. Что представляет собой физическая модель таблицы базы данных?

6. Назовите обязательные характеристики полей таблицы базы данных.

7. Как классифицируются файлы и файловые структуры в базах данных?

8. Что представляет собой файл с позиции пользователя?

9. Назовите типы устройств внешней памяти, которые существуют в зависимости от методов управления доступом к информации в файлах.

10. Назовите характеристики файла, которые хранятся в файловой системе.

11. Что такое хэш-функция и в чем состоит сущность так называемых методов хэширования?

12. В чем отличие индексно-прямых файлов от индексно-последовательных файлов?

13. Какую поисковую структуру принято называть *B*-деревом?

14. На какие четыре вопроса нужно дать ответы, чтобы описать некоторый уровень иерархии памяти?

15. Какие два параметра используются для оценки производительности основной памяти?

16. Что такое виртуальная память и какие актуальные задачи организации обработки (вычислений) большого объема данных были решены с помощью организации виртуальной памяти?

Глава 4

РАЗРАБОТКА И ОРГАНИЗАЦИЯ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

4.1. Базы данных — основа современных CALS-технологий

CALS-технологии — это современное направление развития информационного обеспечения производственных и бизнес-процессов, направленное на создание единого информационного пространства, основу которого составляют интегрированные базы данных.

Аббревиатура CALS расшифровывается двояким образом:

- Continuous Acquisition and Life Support — непрерывная информационная поддержка поставок и жизненного цикла (изделий);
- Commerce At Light Speed — высокоскоростная (быстрая) коммерция.

В обоих случаях речь идет о создании, преобразовании и передаче больших объемов информации между участниками производственных и бизнес-процессов.

Концепция и идеология CALS зародилась в недрах военно-промышленного комплекса США и затем была воспринята всеми странами НАТО.

В России принят адекватный аналог CALS — Информационная поддержка жизненного цикла изделий (ИПИ). На рис. 4.1 представлена схема ИПИ, принятая в России.

В соответствии с данной схемой основу ИПИ составляет интегрированная информационная среда (ИИС), или единое информационное пространство (ЕИП). Оба термина равнозначны, однако в Терминологическом словаре, утвержденном Госстандартом России, принят первый термин.

Данный стандарт определяет ИИС как совокупность распределенных баз данных, содержащих сведения об изделиях, производственной среде, ресурсах и процессах предприятия, обеспечивающая корректность, актуальность, сохранность и доступность данных тем субъектам производственно-хозяйственной деятельности, участвующим в осуществлении жизненного цикла изделия, которым это необходимо и разрешено.

При создании на предприятии ИИС должен реализовываться главный принцип ИПИ: *информация, однажды возникшая на каком-либо этапе производственного процесса, сохраняется и становится доступной всем участникам этого или других этапов в соответствии с имеющимися у них правами пользования этой информацией.*

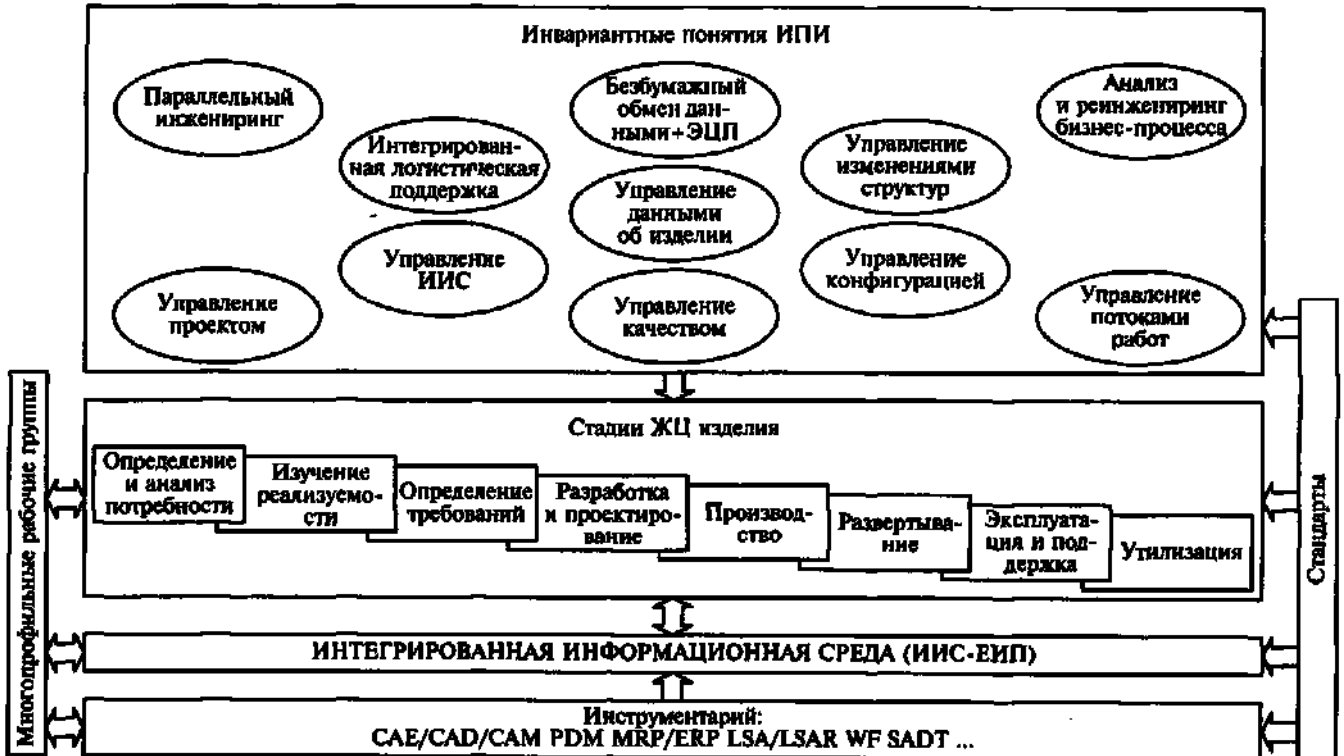


Рис. 4.1. Схема ИПИ

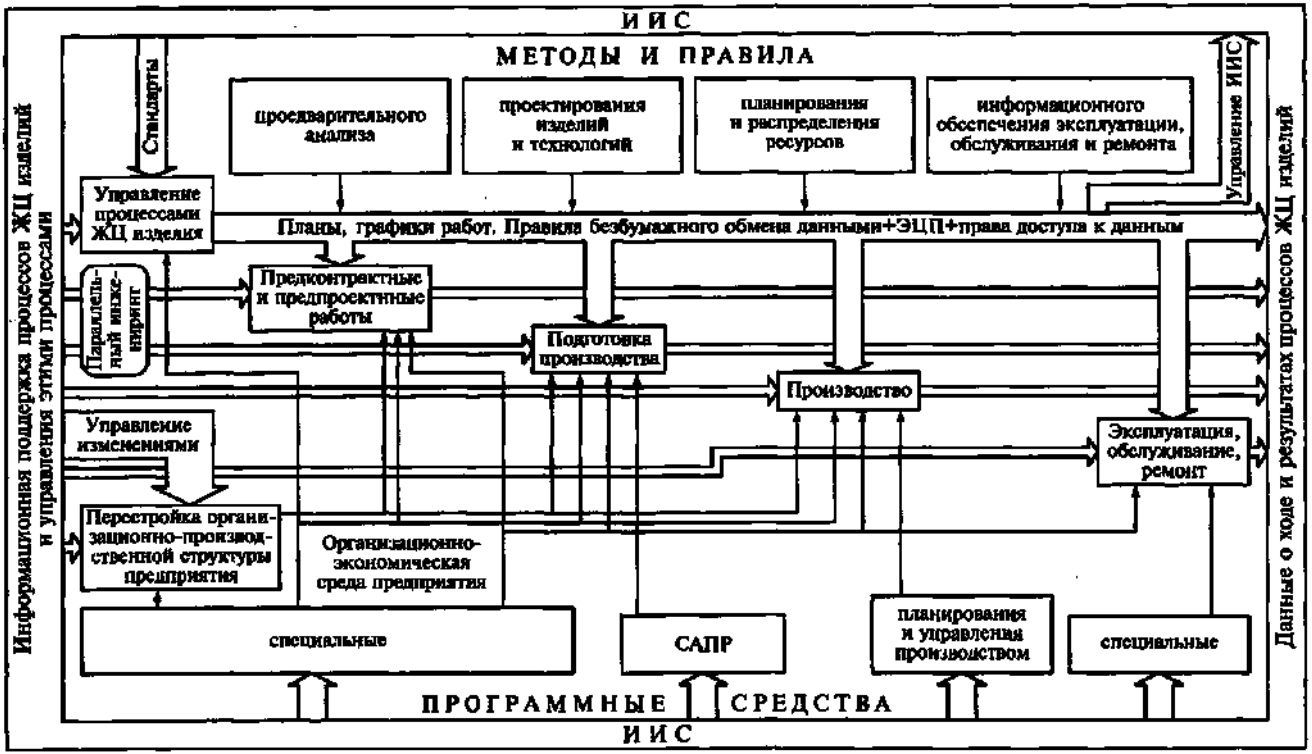


Рис. 4.2. Компоненты формирования общей базы данных промышленного предприятия

Процессы создания, преобразования и передачи информации осуществляются при помощи современных программных средств. К числу таких средств (см. рис. 4.1) относятся:

- системы автоматизированного конструкторского и технологического проектирования (CAE/CAD/CAM);
- программные средства управления данными об изделиях, в том числе СУБД (PDM);
- автоматизированные системы планирования и управления производством (MRP/ERP);
- системы анализа, поддержки и ведения баз данных (LSA/LSAR);
- программные средства управления потоками работ (WF);
- программные средства моделирования и анализа бизнес-процессов (SADT).

Следовательно, основой для создания ЕИП являются базы данных.

Исходя из данной концепции традиционное проектирование базы данных как самостоятельного объекта необходимо существенным образом изменить и перейти к стратегии создания многопользовательских, общих баз данных. На рис. 4.2 представлены некоторые компоненты формирования общей базы данных промышленного предприятия.

4.2. Принципы разработки многопользовательских информационных систем в условиях CALS-технологий

Как следует из концепции CALS-технологий, разрабатываемые на предприятиях информационные системы и базы данных должны быть многопользовательскими.

Принципы разработки многопользовательских баз данных должны сводиться к соблюдению двух обязательных условий: системного подхода и стандартизации.

Системный подход. Системный подход к разработке информационной системы означает, что такая система рассматривается как большая система, состоящая из некоторого множества взаимосвязанных и взаимодействующих между собой элементов. При проектировании информационных систем необходимо соблюдать следующие принципы:

- учет интересов всех потенциальных пользователей систем;
- модульный принцип разработки и внедрения.

Учет интересов всех потенциальных пользователей систем. Этот принцип означает следующую последовательность разработки БД.

1. Установить, каким специалистам и в каких подразделениях предприятия необходима информация о конкретном информационном объекте.

2. Установить признаки описания объектов различными пользователями.

3. Установить общий состав признаков объектов одного класса.

Такой подход к проектированию увеличивает сроки разработки БД, но обеспечивает значительное снижение затрат на разработку всей системы в целом.

Для пояснения этого принципа можно привести следующий реальный пример разработки БД на одном из предприятий. Появление программ создания баз данных было по достоинству оценено сотрудниками и они «бросились» разрабатывать необходимые для себя базы данных.

Одной из задач, стоящих перед технологами цехов, являлась задача выбора инструмента для механической обработки деталей. Они разработали свою, цеховую БД по режущему инструменту (затратив на это и время, и средства).

В то же время в конструкторском отделе завода специалисты, занимающиеся проектированием режущего инструмента, также создали свою БД. Однако когда руководство приняло решение создать общезаводскую информационную систему по режущему инструменту, то оказалось, что одни и те же признаки режущего инструмента разные специалисты описывали разными способами. В результате разработанные базы данных пришлось полностью переделывать, что потребовало как дополнительного времени, так и дополнительных затрат. Средства на разработку несогласованных между специалистами баз данных были потеряны для предприятия.

Модульный принцип разработки и внедрения. Модульный принцип означает, что любая система должна разрабатываться в виде отдельных взаимосвязанных модулей (подсистем), которые могут внедряться в производстве отдельно, до окончательной разработки всей системы.

Стандартизация. Стандартизация разработки информационных систем, учитывая их многопользовательский характер, имеет следующие аспекты:

- информационный;
- программный;
- аппаратный.

Стандартизация *информационного* обеспечения обусловлена принципами компьютерной обработки символьной информации, так как объекты баз данных должны однозначно распознаваться компьютером.

Этот аспект разработки БД означает, что на все информационные объекты должны быть установлены четкие правила их идентификации (грамматические правила их написания). Так, установив название инструмента для механической обработки детали «Резец расточной», не допускается никакого другого способа его

обозначения (например, название «Расточной резец» не идентично названию «Резец расточной»).

Необходимость стандартизации *программного* обеспечения очевидна — при разработке многопользовательских, удаленных друг от друга систем данные одной системы должны обрабатываться программным обеспечением другой системы.

Стандартизация *аппаратного* обеспечения связана с необходимостью снижения затрат на эксплуатацию компьютерной техники.

Внедрение на предприятиях России концепции CALS-технологий предусматривает широчайшее применение единых, в том числе и международных, стандартов.

4.3. Организация многопользовательских систем управления базами данных в локальных вычислительных сетях

Компьютерные информационные системы современных предприятий разрабатываются с применением сетевых технологий — объединением компьютеров в локальные вычислительные сети (ЛВС). При разработке баз данных в локальных вычислительных сетях предприятий применяют два типа (две архитектуры) их организации:

- архитектура файл—сервер;
- архитектура клиент—сервер.

Общими признаками для этих типов организации баз данных является наличие сервера (компьютера), на котором находятся базы (файлы) данных и рабочих станций (компьютеров пользователей) — клиентов.

Отличаются эти две архитектуры организации баз данных способами обработки информации.

В архитектуре файл—сервер все процессы обработки информации производятся на компьютере клиента. Для этого клиенту по соответствующему запросу пересылается весь файл с данными.

В архитектуре клиент—сервер все процессы обработки информации выполняются на сервере по запросу клиента, которому отсылаются только результаты обработки данных.

При организации многопользовательских сетевых баз данных предпочтительным является организация системы по типу клиент— сервер, что следует из недостатков архитектуры файл—сервер и преимуществ архитектуры клиент—сервер.

Недостатки организации БД по архитектуре *файл—сервер*:

- при передаче по сети файлов БД, особенно с большими объемами информации и с учетом возможного обращения к файлам одновременно нескольких пользователей, резко снижается производительность работы с системой;
- при одновременной передаче по сети файлов с большими объемами нескольким пользователям увеличивается вероятность

нарушения достоверности передаваемой информации, что снижает надежность работы системы.

Преимущества организации БД по архитектуре клиент—сервер:

• при передаче по сети только результатов обработки данных по запросам клиентов резко снижается нагрузка на сеть и, как следствие, увеличивается возможность подключения к БД большего числа пользователей. Производительность работы системы значительно выше, чем в архитектуре файл—сервер;

• централизованное хранение и обработка данных на сервере повышает надежность работы системы;

• разработку серверной части СУБД можно выполнять на языке SQL или других языках высокого уровня, что повышает надежность и производительность обработки данных. Разработку клиентской части СУБД можно выполнять с применением прикладных программных продуктов, например Visual Basic, Microsoft Access, что значительно сократит время на разработку информационной системы.

4.4. Этапы проектирования многопользовательских баз данных

В предыдущих подразделах мы установили, что в современных условиях развития производства и бизнеса необходимо перейти от стратегии проектирования баз данных как самостоятельных объектов к стратегии создания многопользовательских информационных систем — общих баз данных. Такой переход предусматривает следующие стадии проектирования многопользовательских баз данных.

1. Разработка концептуальной модели многопользовательской базы данных.
2. Разработка проекта СУБД в соответствии с техническим заданием.
3. Реализация проекта и разработка технической документации.

Разработка концептуальной модели многопользовательской базы данных. На данной стадии проектирования многопользовательских баз данных необходимо выполнить следующие этапы:

• определение цели создания ИИС;

• установление состава пользователей БД;

• разработка концептуальной модели БД;

• разработка технического задания на проектирование локальных СУБД;

• определение потребных трудовых и материальных ресурсов для разработки БД.

Определение цели создания ИИС. Очевидно, что целью разработки любой компьютерной системы является достижение определенного экономического эффекта от ее реализации, поэтому в

условиях конкретного предприятия необходимо установить приоритетные направления в создании ИИС. Базы данных могут разрабатываться практически для всех задач управления производством, например:

- поставка материалов и комплектующих изделий;
- проектирование конструкции новых изделий;
- проектирование технологических процессов изготовления продукции;
- проектирование технологического оснащения (приспособления, инструмент);
- оперативное календарное планирование и управление выпуском изделий;
- разработка нормативной базы (потребность в трудовых и материальных ресурсах, основных и вспомогательных материалах и др.);
- управление качеством выпускаемой продукции;
- управление сбытом и др.

Принятие решения о выборе направления для разработки баз данных, естественно, является прерогативой руководителей предприятия.

Установление состава пользователей БД. Выбрав область производственной деятельности, необходимо установить состав пользователей информацией разрабатываемой базы данных. Это необходимо для решения следующих задач:

- определение классов информационных объектов, их характеристик и, в конечном итоге, определение состава таблиц баз данных;
- определение месторасположения потенциальных пользователей и, в конечном итоге, определение архитектуры ЛВС.

Разработка концептуальной модели БД. Конечной задачей разработки концептуальной модели является установление оптимального состава таблиц базы данных (см. подразд. 3.2). На данном этапе создания многопользовательских баз данных оптимальный состав таблиц определяется сначала исходя из потребностей каждого пользователя ИИС, а затем каждая таблица может быть подвергнута процедуре нормализации.

Разработка технического задания на проектирование локальных СУБД. После определения состава таблиц базы данных и состава пользователей ИИС можно приступить к разработке технического задания на проектирование СУБД. В техническом задании необходимо:

- обосновать выбор архитектуры ЛВС и архитектуры баз данных;
- обосновать выбор программной системы для разработки СУБД;
- разработать требования к формам выходных документов, предоставляющих необходимую информацию для каждого пользователя БД;
- разработать требования к созданию пользовательского интерфейса с учетом задач каждого пользователя;

• разработать требования к организационному обеспечению СУБД, в том числе, определить права доступа пользователей к базе данных и ее компонентам как в процессе заполнения таблиц информацией, так и в процессе получения информации.

Определение потребных трудовых и материальных ресурсов для разработки БД. После выполнения всех перечисленных выше этапов необходимо оценить потребность в трудовых и материальных ресурсах для выполнения задач технического задания.

Для этого целесообразно воспользоваться программной системой управления проектами, например Microsoft Project.

Применение этой системы целесообразно не только для определения потребности в ресурсах; она позволяет эффективно руководить всем ходом выполнения работ по проектированию СУБД. Расширенное семейство продуктов Microsoft Project 2002 сочетает в себе интуитивно-понятные средства управления проектами, доступ к информации и поддержку коллективной работы, а также является мощной платформой корпоративного управления проектами.

Разработав техническое задание и определив состав исполнителей, можно приступить к реализации проекта — созданию системы управления базами данных для выбранного направления производственной деятельности предприятия.

Разработка проекта СУБД в соответствии с техническим заданием. На данной стадии проектирования многопользовательских баз данных необходимо выполнить следующие задачи.

1. Сбор, анализ и подготовка исходной информации об объектах конкретной предметной области для их преобразования в таблицы баз данных.
2. Разработка оптимального состава и структуры таблиц базы данных.
3. Установление логических связей между таблицами.
4. Разработка необходимого числа запросов для реализации поставленной задачи.
5. Разработка необходимого числа отчетов, отвечающих требованиям к выходным документам, определенных техническим заданием.
6. Разработка форм пользовательского интерфейса.
7. Разработка управляющих модулей, автоматизирующих работу пользователя с системой.

Реализация проекта и разработка технической документации. Реализация проекта разработанной СУБД сводится к следующим задачам:

• заполнение таблиц баз данных информацией об объектах;

• проверка функционирования СУБД при выполнении поставленных задач;

• разработка инструкций для пользователей;

• сдача системы заказчику.

4.5. Основные компоненты систем управления реляционными базами данных

Проект СУБД должен содержать, как минимум, следующие основные компоненты:

- таблицы;
- запросы;
- формы;
- отчеты;
- управляющие программы.

Таблицы. Таблицы базы данных могут иметь различное назначение (например, таблицы постоянной информации, таблицы переменной информации).

Таблицы постоянной информации (условно постоянной) должны содержать данные, не меняющиеся в течение длительного времени (например, списки сотрудников организации, названия технологических операций, применяемых при изготовлении продукции, и т.п.).

Таблицы переменной информации — это таблицы, информация об объектах в которых постоянно дополняется или изменяется пользователем.

Запросы. Запросы базы данных представляют собой некоторый набор команд, предназначенных для поиска и обработки информации в таблицах по заданным пользователем условиям (значениям полей). Современные СУБД позволяют формировать запросы:

- на выборку;
- обновление;
- добавление;
- удаление;
- создание таблиц.

Запрос на выборку предназначен для поиска (выбора) информации в конкретной таблице (таблицах) базы данных.

Запросы на обновление предназначены для автоматического обновления данных в отдельных ячейках таблицы.

Запросы на добавление или удаление предназначены для автоматического добавления записей в таблицы или удаления записей из таблиц БД.

Запросы на создание таблиц предназначены для создания новых таблиц на основе уже имеющихся в БД. При этом автоматически формируется структура новой таблицы.

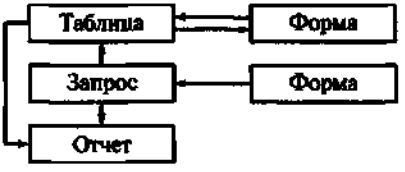


Рис. 4.3. Схема связей элементов СУБД

Формы. Формы при разработке информационных систем предназначены для организации «дру-

жественного» интерфейса между пользователем и компьютером. По назначению формы можно разделить на следующие группы:

- формы для ввода данных в таблицы;
- формы для ввода условий выполнения запросов;
- формы для автоматического управления работой системы (кнопочные формы, формы — меню и др.)

Отчеты. Отчеты — это виды документов для вывода результатов обработки информации. Как правило, отчеты могут соответствовать формам отчетности, принятым на предприятии. Это могут быть формы бухгалтерской отчетности или формы технологической документации и др.

Отчеты разрабатываются на основе информации, содержащейся в таблицах БД или формирующейся в результате выполнения запросов.

При разработке СУБД ее элементы могут быть связаны между собой в соответствии со схемой, представленной на рис. 4.3.

Управляющие программы. Управляющие программы предназначены для автоматизации работы с компонентами базы данных. Они пишутся с помощью макрокоманд (макросов) или на языке программирования, например VBA.

Контрольные вопросы

1. Дайте определение основному направлению современного совершенствования производства и бизнеса — CALS-технологии.
2. Что означают следующие принципы разработки многопользовательских систем управления базами данных: учет интересов всех потенциальных пользователей систем и модульный принцип разработки и внедрения системы?
3. Назовите основные этапы проектирования многопользовательских баз данных.
4. Какие задачи необходимо решить при разработке проекта базы данных?
5. Назовите основные компоненты систем управления реляционными базами данных.
6. Назовите основные характеристики, достоинства и недостатки следующих форм организации многопользовательских баз данных: архитектура файл — сервер и архитектура клиент — сервер.

Глава 5

ОБЗОР ПРОГРАММНЫХ ПРОДУКТОВ ДЛЯ РАЗРАБОТКИ СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

5.1. История развития программных средств разработки баз данных

На ранних стадиях разработки информационно-поисковых систем разрабатывались специальные языки манипулирования данными (ЯМД) — языки запросов. Они были ориентированы на операции с данными, представленными в виде иерархически связанных файлов, и имели соответствующие алгоритмы поиска информации.

Появление реляционных баз данных создало предпосылки для других, более быстрых алгоритмов поиска информации.

Для обработки информации, структурированной в виде таблиц — двумерных массивов, в конце 70-х гг. XX в. фирмой IBM был разработан соответствующий язык, который в дальнейшем получил название Structured Query Language (SQL) — язык структурированных запросов. В настоящее время SQL является международным стандартом языка обработки данных в реляционных СУБД. Язык SQL является ядром всех программных продуктов для разработки СУБД.

Набольшее распространение среди пользователей и разработчиков СУБД получили следующие программные продукты:

- специальные языки программирования — Visual FoxPro, SQL, MS SQL-Server;
- прикладные программные системы — Microsoft Access, Oracle и др.

Рассмотрим некоторые характеристики данных программных средств.

Visual FoxPro. Этот язык программирования представляет собой дальнейшее развитие одного из популярных языков разработки баз данных — FoxPro. Принципиальным отличием Visual FoxPro от его «прародителя» FoxPro является возможность «визуального» — объектно-ориентированного программирования практически всех компонентов СУБД. Интерфейс Visual FoxPro полностью соответствует графической оболочке операционных систем Windows, что делает работу по созданию СУБД достаточно понятной для тех, кто имеет на своих компьютерах данные операционные системы.

Несмотря на наличие средств «визуального программирования» мы рекомендуем эту программную систему для программистов. Возможности Visual FoxPro предполагают разработку локальных или

многопользовательских баз данных в пределах одного предприятия.

MS SQL-Server. Данная программная система в основном предназначена не для разработки пользовательских приложений, а для управления многопользовательскими базами данных, разработанными по архитектуре клиент — сервер. Эта система позволяет управлять базами данных (тиражировать данные, вести их параллельную обработку, получать и передавать данные как в локальной вычислительной сети предприятия, так и через сеть Интернет и др.), взаимодействуя с клиентскими компьютерами, имеющими различные по техническим характеристикам аппаратные средства. SQL-Server предназначен для обработки значительных объемов информации, но, как правило, не более терабайт, что вполне достаточно для отдельных предприятий.

Microsoft ACCESS. Это одна из самых популярных прикладных программных систем для разработки баз данных.

Microsoft ACCESS — это программная среда, разработанная фирмой Microsoft. Она предназначена для создания систем управления реляционными базами данных с достаточно большими объемами информации (сотни мегабайт). Microsoft ACCESS предоставляет пользователю все необходимые средства для автоматизации создания и обработки данных, а также для управления данными при работе.

Основным достоинством данной системы является ее ориентация не на программиста, а на конечного пользователя.

Последние версии Microsoft Access позволяют применять ее для создания многопользовательских баз данных. В этом случае таблицы баз данных могут быть переданы на сервер, а пользовательский интерфейс сохранен на компьютере клиента. В этом случае представляется возможным сочетать простоту разработки всех компонентов СУБД с применением Microsoft Access, а задачи управления многопользовательскими базами данных возложить на MS SQL-Server.

Другим достоинством Microsoft Access является ее неоспоримое преимущество перед всеми другими программными продуктами в качестве средства для обучения разработке баз данных.

Oracle. Эта система предназначена для разработки корпоративных реляционных баз данных, объемы информации в которых превышают терабайты. Основу системы составляет также язык SQL. Oracle отличается возможностью высокой степени защиты данных.

5.2. Структурированный язык запросов SQL

Структурированный язык запросов SQL является обычным языком программирования, состоящим из операторов и правил грамматики. Запрос к таблице базы данных на языке SQL представляет

Таблица 5.1

Операторы определения данных

Оператор	Действие
CREATE TABLE	Создает новую таблицу БД
DROP TABLE	Удаляет таблицу из БД
ALTER TABLE	Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы
CREATE VIEW	Создает виртуальную таблицу, соответствующую некоторому SQL-запросу
ALTER VIEW	Изменяет ранее созданное представление
DROP VIEW	Удаляет ранее созданное представление
CREATE INDEX	Создает индекс для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс
DROP INDEX	Удаляет ранее созданный индекс

собой инструкцию SELECT, которую можно описать следующим образом.

SELECT [ALL] (Список полей таблицы или запроса)
FROM (Список таблиц или запросов, на основе которых формируется запрос)

Таблица 5.2

Операторы манипулирования данными

Оператор	Действие
DELETE	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно
INSERT	Вставляет одну строку в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
UPDATE	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации

Таблица 5.3

Оператор запросов

Оператор	Действие
SELECT	Оператор, заменяющий все операторы реляционной алгебры и позволяющий сформировать результирующее отношение, соответствующее запросу

Таблица 5.4

Операторы управления действиями (транзакциями)

Оператор	Действие
CCOMMIT	Завершает комплексную, взаимосвязанную обработку информации, объединенную в транзакцию
ROLLBACK	Отменяет изменения, проведенные в ходе выполнения транзакции
SAVEPOINT	Сохраняет промежуточное состояние БД, помечает его для того, чтобы можно было в дальнейшем к нему вернуться

Таблица 5.5

Операторы администрирования данными

Оператор	Действие
ALTER DATABASE	Изменяет набор основных объектов в базе данных, ограничений, касающихся всей базы данных
ALTER DBAREA	Изменяет ранее созданную область хранения
ALTER PASSWORD	Изменяет пароль для всей базы данных
CREATE DATABASE	Создает новую базу данных
CREATE DBAREA	Создает новую область хранения базы данных
DROP DATABASE	Удаляет базу данных
DROP DBAREA	Удаляет область хранения базы данных
GRANT	Предоставляет права доступа к базе данных или отдельным ее элементам
REVOKE	Лишает права доступа к базе данных или отдельным ее элементам

Таблица 5.6

Операторы управления курсором

Оператор	Действие
DECLARE	Определяет курсор для запроса. Задает имя и определяет связанный с ним запрос к БД
OPEN	Открывает курсор. Открывает объект базы данных
FETH	Устанавливает курсор на определенную запись и считывает ее
CLOSE	Закрывает курсор. Закрывает объект базы данных
PREPARE	Генерирует план выполнения запроса в соответствии с инструкцией SELECT
EXECUTE	Выполняет сгенерированный ранее запрос

[WHERE (Условия отбора данных)]
[GROUP BY (Список полей, выводимых в результат выполнения запроса)]
[HAVING (Условия для группировки данных в запросе)]
[ORDER BY (Список полей, по которым упорядочивается вывод данных в запросе)]

В рассмотренной структуре инструкции SELECT ALL — ключевое слово, которое означает, что в результирующий набор записей включаются все записи таблицы или запроса, которые удовлетворяют условиям запроса.

Ключевые слова могут отсутствовать в запросе.

В зависимости от характера выполняемых действий операторы SQL можно разделить на следующие группы:

- операторы определения данных;
- операторы манипулирования данными;
- операторы (язык) запросов;
- операторы управления действиями (транзакциями);
- операторы администрирования данными;
- операторы управления (управления курсором).

В табл. 5.1 — 5.6 представлены соответствующие группы операторов языка SQL и выполняемые ими действия.

5.3. Общие сведения об MS SQL Server 7.0

Корпорация Microsoft, которая является всемирно признанным лидером в разработке программного обеспечения, провела обширные исследования рынка баз данных и требований, предъявляемых к современным системам управления базами дан-

ных. На основе этой информации была разработана стратегия для создания нового продукта корпорации из семейства SQL Server. Новая версия системы управления базами данных SQL Server 7.0 является действительно современным продуктом, вобравшим в себя множество новейших технологий. Особенностью SQL Server 7.0 является то, что он предназначен как для локальных баз данных, так и для баз данных масштаба предприятия, имеющих десятки таблиц, сотни пользователей и миллионы строк данных. Причем в обоих случаях используется один и тот же программный код. SQL Server 7.0 может работать как под управлением операционной системы Windows NT, так и под управлением Windows 95/98.

В седьмой версии SQL Server реализовано множество новых функциональных возможностей, включая новый способ организации хранения данных, новые утилиты администрирования, поддержку технологий OLE DB 2.0, ActiveX, COM, реализацию технологий Microsoft Search, OLAP, Microsoft Repository и другие изменения. Большинство современных систем управления базами данных сложны в администрировании и предназначены для пользователей, имеющих специальную подготовку и опыт работы. SQL Server 7.0 является продуктом, масштабируемым от ноутбука до многопроцессорного кластера. Он вобрал в себя множество новейших решений и технологий, позволяющих облегчить бремя администратора и упростить процесс разработки и сопровождения баз данных.

Максимальное число байт в строке достигает 8060, а таблица может иметь до 1024 колонок. Появились новые типы данных, реализована поддержка Unicode, улучшилась система блокировок. Диспетчер блокировок оптимизирован, теперь он выполняет запросы на блокировку быстрее и с меньшими внутренними потерями на синхронизацию.

Динамическое самоуправление SQL Server. SQL Server 7.0 облегчает администрирование сервера, обеспечивая для некоторых параметров конфигурации режим автоматического конфигурирования. Сервер постоянно отслеживает потребность в тех или иных ресурсах и динамически изменяет параметры своей настройки. Например, если одна из баз данных больше не используется и автоматически закрывается сервером, то требования к оперативной памяти и процессорному времени снижаются. При использовании статических значений для конфигурирования сервера неиспользуемые ресурсы все равно будут зарезервированы операционной системой для SQL Server и не могут быть использованы другими приложениями. В режиме автоматического конфигурирования SQL Server будет возвращать неиспользуемые ресурсы операционной системе, что увеличит производительность как самой системы, так и прикладных программ.

В результате выполнения операций вставки и удаления объем памяти, занимаемый базой данных, постоянно меняется. Диспетчер блокировок SQL Server 7.0 динамически управляет объемом используемых им ресурсов при работе с большими базами данных, что избавляет администратора от необходимости вручную изменять параметры конфигурирования сервера.

Возможность автоматического конфигурирования особенно полезна при реализации SQL Server на небольших системах, таких как ноутбук. При этом пользователь может более продуктивно работать с продуктом, не беспокоясь о конфигурировании сервера. Если вы чувствуете себя достаточно опытным администратором, который не боится трудностей, то можете отказаться от автоматического управления SQL Server. В этом случае вы получите полный контроль над конфигурированием системы.

Обработчик запросов. В обработчике запросов SQL Server 7.0 были реализованы новые методы поиска, повышающие скорость обработки комплексных запросов. В отличие от предыдущей версии, в которой использовался единственный метод соединения при помощи вложенных циклов, обработчик запросов SQL Server 7.0 использует методы реляционного соединения одинаково хэшированных отношений (hash join), реляционного соединения отсортированных отношений слиянием (merge join) и агрегирования на основе хэширования (hash aggregation). Причем внутри одного запроса могут использоваться различные методы соединения. Перед тем как приступить к извлечению данных из таблиц, их можно предварительно отфильтровать. Это стало возможным благодаря тому, что SQL Server использует технику пересечения и объединения индексов для таблиц с несколькими индексами. Все индексы для одной таблицы ведутся одновременно, а учет ограничений включается в план выполнения запроса.

SQL Server поддерживает параллельное выполнение запросов. Если сервер имеет несколько процессоров, то выполнение запроса будет равномерно распределено между ними. SQL Server сам решает, когда параллельное выполнение запроса приведет к увеличению скорости его обработки, и составляет план выполнения запроса.

В новой версии SQL Server реализована поддержка распределенных транзакций, что позволяет включать в один запрос удаленные серверы. Обработчик запросов использует технологию OLE DB для взаимодействия с другими источниками данных. Это обеспечивает более тесную интеграцию с другими продуктами Microsoft и облегчает процесс разработки приложений для SQL Server. Еще одним преимуществом использования технологии OLE DB является применение ее для доступа к не реляционным источникам данных. При этом реализуется новая технология, называемая Universal Access (универсальный доступ), которая по-

зволяет получать доступ к данным в контексте одного запроса. Это делает ненужным применение так называемых универсальных серверов для промежуточного хранения экспортируемых данных.

Поддержка баз данных больших объемов. Предыдущие версии SQL Server поддерживали базы данных объемом до 300 Мбайт. SQL Server 7.0 может работать с базами объемом в несколько терабайт. В этой версии реализована поддержка стандарта записи на магнитную ленту Microsoft Tape Format. Это позволяет использовать общее устройство резервного копирования для хранения архивов SQL Server и архивов операционной системы. Изменения коснулись и самого процесса резервного копирования содержимого баз данных. Во-первых, операции резервного копирования и восстановления данных выполняются намного быстрее. Во-вторых, резервное копирование архивирует только те страницы, которые были созданы либо подверглись изменениям с момента выполнения последнего полного резервного копирования. Кроме того, можно также выполнять резервное копирование отдельных файлов и файловых групп, что позволяет разбить процесс архивации всей базы данных на несколько этапов.

Система безопасности SQL Server. В новой версии SQL Server реализуется более плотная интеграция с Windows NT. Для этого можно создавать роли, включая в них не только учетные записи Windows NT, но и собственные учетные записи SQL Server. Пользователь может быть участником многих ролей, что позволяет более эффективно управлять доступом к отдельным объектам баз данных. Появились новые стандартные роли, обеспечивающие более гибкое распределение полномочий среди персонала, обслуживающего сервер.

Инструменты администрирования SQL Server. Инструменты администрирования SQL Server 7.0 предоставляют широкие возможности для управления серверами баз данных, оптимизации запросов и разрешения возникающих проблем.

SQL Server Enterprise Manager. В новой версии SQL Server административный инструмент Enterprise Manager реализован в виде подгружаемого модуля Microsoft Management Console. Он позволяет управлять всеми серверами баз данных в сети предприятия, представляя все объекты SQL Server в виде иерархического дерева.

Используя Enterprise Manager, можно управлять настройками различных серверов и настройками баз данных, конфигурировать систему безопасности. Можно создавать и изменять таблицы, представления, полнотекстовые индексы, хранимые процедуры, назначать операторов, управлять оповещеньями, что очень важно при организации на предприятии системы CALS-технологий. Кроме того, Enterprise Manager позволяет создавать задания. В SQL Server

7.0 задание может включать в себя множество шагов, каждый из которых задается:

- средствами Transact-SQL;
- с помощью Microsoft Visual Basic Scripting Edition;
- используя Microsoft Jscript;
- командами операционной системы.

Большинство действий по обслуживанию SQL Server удобнее всего выполнять, используя Enterprise Manager. Можно также управлять конфигурацией сервера, используя хранимые процедуры или обращаясь к системным таблицам напрямую. Любое действие в Enterprise Manager можно выполнить двумя способами:

- командами контекстного меню, попасть в которое можно, указав на объект и нажав правую кнопку мыши;
- использованием мастеров (wizards).

SQL Server Service Manager. Утилита Service Manager предназначена для управления работой следующих служб SQL Server:

- MSSQLServer — служба, которая запускает SQL Server;
- SQLServerAgent — служба, отвечающая за автоматическое выполнение задач администрирования;
- MSDTC — служба, управляющая выполнением распределенных транзакций;
- MSSearch — служба, с помощью которой реализуется полнотекстовый поиск.

С помощью Service Manager можно запускать, приостанавливать и останавливать перечисленные службы на любом SQL Server в сети, а также разрешать или запрещать их автоматический запуск при старте операционной системы.

SQL Server Performance Monitor. После установки SQL Server 7.0 можно использовать стандартную утилиту диагностирования Windows NT — Performance Monitor — для сбора информации о работе СУБД. Утилита Performance Monitor собирает в основном статистическую информацию, такую как число транзакций в единицу времени, число открытых соединений или число выполненных запросов. Для более детального контроля используется утилита Profiler, поставляемая с SQL Server 7.0.

SQL Server Profiler. Утилита Profiler предназначена для детального анализа работы SQL Server. Используя эту утилиту, можно собирать информацию о времени выполнения запросов и хранимых процедур, об установленных соединениях, установленных блокировках, активных транзакциях и о многом другом.

SQL Server Query Analyzer. В SQL Server 7.0 для отладки и оптимизации запросов используется графическая утилита Query Analyzer. Она предоставляет пользователю следующие возможности:

- встроенный текстовый редактор, в котором можно набирать команды Transact-SQL;

Таблица 5.7

Мастера SQL Server

Название	Описание
Backup Wizard	Выполняет резервное копирование базы данных
Failover Setup Wizard	Помогает организовать кластер на основе SQL Server
Configuring Publishing and Distribution Wizard	Облегчает процесс конфигурации издателя и дистрибьютора при репликации
Create Alert Wizard	Создает оповещение
Create Database Wizard	Создает базу данных
Create Diagram Wizard	Создает диаграмму базы данных
Create Index Wizard	Создает индекс
Create Job Wizard	Создает задание
Create New DataSource Wizard	Инсталлирует ODBC-драйвер и ODBC — источник данных
Create Login Wizard	Создает учетную запись SQL Server для пользователя
Create Publication Wizard	Создает публикацию для последующей репликации
Create Stored Procedure Wizard	Создает хранимую процедуру
Create Trace Wizard	Создает трассировку для Profiler
Create View Wizard	Создает представление
Create Maintenance Plan Wizard	Создает файл поддержки
Disable Publishing and Distribution Wizard	Удаляет издателя и дистрибьютора для репликации
Full-text Indexing Wizard	Определяет полнотекстовые индексы
Index Tuning Wizard	Оптимизирует индексы
Make Master Server Wizard	Устанавливает мастер-сервер
Make Target Server Wizard	Устанавливает сервер-приемник
Register Server Wizard	Облегчает процесс регистрации серверов в Enterprise Manager
Pull Subscription izard	Конфигурирует подписчика для вытягивания данных

Окончание табл. 5.7

Название	Описание
Push Subscription Wizard	Конфигурирует подписчика с выталкивающим издателем
SQL Server Upgrade Wizard	Позволяет обновить базы данных
Web Assistant Wizard	Создает Web-задачи
DTS Import Wizard	Создает DTS — пакет для импорта данных в SQL Server

• различные команды Transact-SQL отображаются разным цветом, что позволяет легко ориентироваться в сложных запросах;

• логические шаги при выполнении запроса отображаются в виде графической диаграммы. Это позволяет определить, какая именно часть запроса использует ресурсы неоптимальным образом;

• мастер оптимизации индексов помогает определить, позволит ли введение добавочных индексов увеличить производительность запроса.

SQL Server Upgrade Wizard. При помощи мастера Upgrade Wizard можно преобразовать базы данных SQL Server 7.0 для использования в новой версии. При этом происходит обновление не только самой базы данных, но и большинства настроек сервера, таких как настройки репликации.

SQL Server включает в себя множество мастеров, которые призваны облегчить администратору выполнение повседневных задач. В табл. 5.7 приведен список имеющихся мастеров.

5.4. СУБД Microsoft Access

Microsoft ACCESS — это программная среда, разработанная фирмой Microsoft. Она предназначена для создания систем управления реляционными базами данных с достаточно большими объемами информации (сотни мегабайт). Microsoft ACCESS предоставляет пользователю все необходимые средства для автоматизации создания и обработки данных, а также для управления данными при работе.

Система Microsoft Access имеет достаточные возможности для разработки информационных систем в различных сферах производства и бизнеса. Хотя ее называют настольной СУБД, она обладает такими характеристиками, которые позволяют применять ее не только для разработки «настольных» информационных систем, но и для таких сложных многопользовательских задач, как разработка систем автоматизированного проектирования или разработка

автоматизированных экспертных систем. (В таблице, разработанной с применением Access 2000 может содержаться до 2 млрд записей.)

Например, фирма ТОП-СИСТЕМЫ разработала с применением Microsoft Access систему автоматизированного проектирования технологических процессов изготовления деталей — ТЕХНО-ПРО, интегрированную в систему параметрического черчения Tflex-CAD.

На кафедре «Технологии производства приборов и систем управления летательных аппаратов» МАТИ — РГТУ им. К.Э. Циолковского разработаны САПР технологических процессов сборки «ЛАЗЕР-2000», экспертная система «КЛАСС-ЭКСПЕРТ» для оценки ожидаемых затрат на изготовление новых приборов.

СУБД ACCESS обладает следующими характеристиками, ставящими ее вне конкуренции с другими системами для обучения методологии разработки информационных систем на основе баз данных:

- простота освоения специалистами, не владеющими языками программирования, что сокращает время на проектирование и уменьшает затраты на разработку системы;
- совместимость с приложениями Windows;
- возможность создания БД со вставкой графических и мультимедийных объектов;
- возможность работы в локальных и глобальных сетях.
- возможность использования таблиц БД, разработанных другими программными системами.

Одним из достоинств СУБД ACCESS является возможность работы с данными, разработанными с применением других программных продуктов, возможность импортировать данные в другие базы данных или экспортировать данные из других баз данных.

ACCESS взаимодействует с любыми базами данных, поддерживающими стандарт открытого доступа к данным ODBC (Open Database Connectivity).

Все действия с информацией в СУБД ACCESS — чтение, вставка, удаление данных и другие — выполняются командами языка SQL.

Контрольные вопросы

1. Назовите программные продукты для разработки баз данных и дайте им сравнительные характеристики.
2. Какую роль играет язык SQL в программных системах разработки баз данных?
3. Назовите назначение следующих слов языка:
   - FROM;
   - WHERE;

- GROUP BY;
- HAVING;
- ORDER BY.

4. Из каких групп операторов состоит язык SQL?
5. Что является основной отличительной особенностью системы SQL Server 7.0. по сравнению с предыдущими версиями?
6. Назовите основные задачи, решаемые с помощью следующих инструментов SQL Server 7.0.:
   - SQL Server Enterprise Manager;
   - SQL Server Query Analyzer;
   - SQL Server Upgrade Wizard.
7. Какие функции выполняют следующие Мастера SQL Server 7.0:
   - Create Database Wizard;
   - Web Assistant Wizard;
   - Create Alert Wizard.
8. Назовите области применения системы Microsoft Access.

стовое поле могут вводиться и одни цифры, если с ними не предполагается производить вычислений.

MEMO. Поле *MEMO* называют полем текстовых примечаний. Этот тип поля предназначен для ввода в него текстовой информации длиной более 255 символов (в Access 2000 — до 65 535 символов). Этот тип данных отличается от текстового тем, что в таблице хранятся не сами данные, а ссылки на блоки данных, которые хранятся отдельно. Это значительно ускоряет обработку таблиц. Поля *MEMO* не могут быть ключевыми или индексными.

Числовой. Этот тип данных для характеристик объектов базы данных, которые могут участвовать в математических расчетах.

Дата/Время. Такой тип данных предназначен для указания даты или времени, характеризующих конкретную запись таблицы (например, дата поступления товара на склад или время начала и окончания работы пользователя в сети Интернет). В данное поле можно вводить даты с 100 по 9999 г.

Денежный. Этот тип данных аналогичен числовому. Отличается от него только характеристиками вводимых чисел. Точность числа не превышает четырех знаков после запятой. Целая часть может содержать до 15 десятичных разрядов. В конце числа могут быть проставлены обозначения валюты (р. или $).

Счетчик. Поле содержит уникальный (не повторяющийся) номер записи таблицы БД. Значения этого поля не обновляются.

Логический. Тип поля, параметры которого могут принимать только два значения, интерпретированные как ДА или НЕТ (Да/Нет), Истина/Ложь, Включено/Выключено. Поля логического типа не могут быть ключевыми, но могут быть индексными.

OLE (*OLE-объект*). В ячейки поля данного типа вводятся ссылки на приложения, разработанные для Windows. Это могут быть текстовые, графические и мультимедийные файлы. Объем хранимых данных в ячейках данного поля ограничен только дисковым пространством компьютера.

Гиперссылка (*Hyperlink*). Этот тип данных позволяет вставлять в поле гиперссылку, с помощью которой можно сослаться на любой файл или фрагмент файла, находящегося на том же компьютере, на котором находится таблица БД, или на любом компьютере в локальной сети или сети Интернет. Гиперссылка состоит из трех частей: адрес, указывающий путь к файлу; дополнительный адрес, указывающий положение фрагмента внутри файла или страницы текста; отображаемый текст. Каждая часть гиперссылки может содержать до 2048 символов.

Мастер подстановок. При выборе этого типа имеется возможность создать фиксированный список значений, которые могут принимать данные, заносимые в ячейки поля.

После установления имени и типа данных следует поместить курсор в соответствующую строку блока *Описание* и ввести ком-

ментарий, позволяющий пользователю правильно вводить информацию при заполнении таблицы.

Мы рекомендуем обязательно вводить комментарий, особенно для тех случаев, когда в обозначении имени или подписи поля содержится недостаточно информации для правильного ввода данных.

После ввода комментария необходимо перейти к блоку *Свойства поля*, разделу *Общие* и задать полю необходимые свойства. В *Конструкторе таблиц* каждому полю в зависимости от типа данных автоматически (по умолчанию) задается определенный набор свойств. Конструируя таблицу, эти свойства можно изменять в соответствии с конкретными требованиями к данным.

В табл. 6.2 перечислены характеристики свойств полей, задаваемые в информационном блоке *Свойства поля*, *Общие*.

После описания характеристик (свойств) всех полей таблицы конструктор закрывают; при этом открываются диалоговые окна, в которых предлагается задать имя таблицы и установить ключевые поля, если они не были заданы.

Имя таблицы. При задании имени таблицы необходимо учесть следующие рекомендации:

- имя поля должно отражать содержание данных в таблице (класс объектов);
- в имени таблицы не должно быть знаков препинания, скобок, восклицательных знаков;
- имя таблицы не должно начинаться с пробела;
- в одном файле БД не должно быть таблиц с одинаковыми именами.

Таблица 6.2

Характеристики полей таблиц БД

Свойство поля	Характеристика
Размер поля	Устанавливает максимальный размер данных, вводимых в ячейки данного поля. Размер данных текстовых (символьных) полей не может превышать 255 знаков. Для числовых полей размер водимых данных устанавливается автоматически в зависимости от типа числа: байт — целые числа от 0 до 255 — 1 байт; целое — целые числа от –32 768 до +32 767 — 2 байт; длинное — целые числа от –2 147 483 648 до +2 147 483 648; целое с плавающей точкой с точностью до шести знаков — числа от –3,4 × 1038 до +3,4 × 1038 — 4 байт; целое с плавающей точкой с точностью до восьми знаков — числа от –1,797 × 10 308 до +1,797 × 10 308 — 8 байт

Продолжение табл. 6.2

Свойство поля	Характеристика
Формат поля	Для полей типа *Текстовый* и *MEMO* можно задавать формат ввода данных, в соответствии с которым данные будут выводиться на экран дисплея. Для полей типов *Числовой*, *Денежный* могут быть выбраны следующие форматы: стандартный — формат, устанавливаемый по умолчанию (отсутствуют разделители тысяч, знаки валют, число десятичных знаков соответствует точности числа); денежный — устанавливается два знака после запятой и выводится символ валюты; фиксированный — как минимум, один знак до запятой и два знака после запятой; с разделителями тысяч — два знака после запятой и разделитель тысяч; процентный — в конце числа выводится знак процента; экспоненциальный — числа выводятся в экспоненциальном виде (например, 1.10 × 103). Для полей типа *Дата/Время* существуют следующие форматы: полный формат даты — устанавливается по умолчанию и имеет, например, следующий вид: 15.04.97.05:30:10 PM; длинный формат даты, например: Пятница, 13 апреля 1997; средний формат даты, например: 13-апр-97; краткий формат даты, например: 13.04.97; длинный формат времени, например: 14:33:10; средний формат времени, например: 14:33 PM; краткий формат времени, например: 14:33. Для полей логического типа могут применяться следующие форматы: Да/Нет; Истина/Ложь; Вкл/Выкл
Число десятичных знаков (точность поля)	Задается для полей типов *Числовой* и *Денежный*. Число знаков — от 0 до 15
Маска ввода	Маска устанавливает шаблон для ввода данных в поля типов *Текстовый*, *Числовой*, *Денежный*, *Дата/Время*. Маска ввода для полей типа *Дата/Время* соответствует выбранному формату

Продолжение табл. 6.2

Свойство поля	Характеристика
Подпись поля	Предназначена для более описательного названия поля, которое будет вводиться в заголовки («шапки») таблиц и другие элементы форм, отчетов. Если подпись поля не вводится, то в соответствующих элементах таблиц, форм и отчетов будут вводится имена полей
Условие на значение	Устанавливает ограничения на значения вводимых данных. Например, задание условия «<100» для числового поля означает, что в это поле нельзя вводить данные более 100. Условие вида «Москва» OR «Вологда» OR «Новосибирск» означает, что вводимые названия городов должны быть только Москва, или Вологда, или Новосибирск. Условия на значение вводимых данных задаются выражениями, состоящими из операторов сравнения, и значениями, которые используются для сравнения. При задании условий применяются известные операторы: < (меньше); <= (меньше или равно); > (больше); >= (больше или равно) При задании условий применяются известные операторы: = (равно); <> (не равно). В выражениях могут применяться логические операторы: OR (или), AND (и), а также операторы сравнения: BETWEEN, IN, LIKE: BETWEEN — проверяет, что введенное значение поля находится внутри заданного диапазона. Верхняя и нижняя границы диапазона разделяются логическим оператором AND. Например, выражение BETWEEN 20 AND 45 означает, что вводимое значение должно находиться в интервале от 20 до 45. Это выражение также может быть записано в виде: >50 AND < 100; IN — проверяет равенство введенного значения поля любому значению из заданного списка. Например, IN («Москва», «Вологда», «Новосибирск») означает, что это выражение соответствует также выражению «Москва» OR «Вологда» OR «Новосибирск»; LIKE — проверяет соответствие полей *Текстовый* или *Мемо* заданному шаблону символов. Например, выражение LIKE «Тех*» означает, что вводимая строка символов должна начинаться с символов «Тех»

Структуру описания запроса на языке SQL рассмотрим на следующем примере.

```
SELECT Нагрузка.фио, Нагрузка.Дол, Нагрузка.Семестр, Нагрузка.Предмет, Нагрузка.Группа, Нагрузка.Лекции, Нагрузка.Конс, Нагрузка.Лабы, Нагрузка.ПЗ, Нагрузка.КРП, Нагрузка.ИР, Нагрузка.Экз, Нагрузка.Зач, Нагрузка.Др_вид, [Лекции] + [Конс] + + [Лабы] + [ПЗ] + [КРП] + [ИР] + [Экз] + [Зач] + [Др_вид] AS Итого, [Итого]/568.639 AS Ставка
FROM Нагрузка
WHERE (Нагрузка.фио) = [Forms]![Нагрузка кафедры]![Поле СоСписком0]));
```

Инструкция (оператор) SELECT является ядром языка. Она применяется для выбора полей из таблицы БД. В данном примере перечислены все поля таблицы «Нагрузка», которые были введены в запрос.

Предложение FROM является частью инструкции и служит для определения источника данных запроса (таблицы или запроса). В данном случае — это таблица «Нагрузка».

Предложение WHERE устанавливает условия отбора данных при выполнении запроса. В данном предложении указано, что условием отбора данных является значение поля *ФИО*, которое равно значению, введенному в *ПолеСоСписком0* формы *Нагрузка кафедры*.

В общем виде синтаксис инструкции SELECT можно описать следующим образом.

```
SELECT [ALL] (Список полей таблицы или запроса)
FROM (Список таблиц или запросов, на основе которых формируется запрос)
[ WHERE (Условия отбора данных)]
[GROUP BY (Список полей, выводимых в результат выполнения запроса]
[HAVING (Условия для группировки данных в запросе]
[ORDER BY (Список полей, по которым упорядочивается вывод данных в запросе]
```

В рассмотренной структуре инструкции SELECT ALL — ключевое слово, которое означает, что в результирующий набор записей включаются все записи таблицы или запроса, которые удовлетворяют условиям запроса.

Ключевые слова могут отсутствовать в запросе.

Контрольные вопросы

1. Из каких информационных блоков состоит *Конструктор таблиц* и в какой последовательности их следует заполнять?
2. Из скольких символов может состоять имя поля?
3. Может ли имя поля начинаться с пробелов?

4. Какие символы не допускаются при обозначении имени поля?

5. В чем состоит отличие текстового типа данных от МЕМО?

6. В чем состоит отличие числового типа данных от денежного?

7. В каких случаях следует применять тип данных OLE?

8. В каких случаях следует применять тип данных *Гиперссылка*?

9. В каких случаях полю присваивают свойство *Ключевое поле*?

10. Может ли ключевое поле иметь повторяющиеся значения данных в таблице БД?

11. В каких случаях полю присваивают свойство *Обязательное*?

12. Какие таблицы называются главными, а какие — подчиненными?

13. Какой смысл имеет термин «Обеспечение целостности данных»?

14. Назовите назначение и виды запросов, разрабатываемых в СУБД ACCESS.

15. В чем состоит отличие постоянного запроса от параметрического?

16. Каково назначение перекрестного запроса?

17. Назовите типы запросов по выполняемым действиям.

18. Назовите правила ввода условий отбора данных в текстовые поля.

19. В чем состоит различие между условиями отбора данных, связанных отношениями AND и OR?

20. Каково назначение следующих функций: Day; Month; Year; Date()?

21. В каких случаях в запросах создается расчетное поле?

22. Какова последовательность действий при создании расчетного поля в запросе с применением построителя выражений?

Продолжение табл. 7.1

Макрокоманда	Назначение
ЗапускПрограммы	Выполняет процедуру Access BASIC
Запуск Макроса	Запускает другой макрос
ОстановитьВсе Макросы	Останавливает работу всех макросов, включая текущий
ОстановитьМакрос	Останавливает работу текущего макроса
Установка значений	
Обновление	Эта команда может быть использована для обновления данных в активном объекте (в режиме *Формы* или *Таблицы*)
Команды Клавиатуры	Запоминает в буфере обмена последовательность нажимаемых клавиш
ЗадатьЗначение	Обновляет или изменяет значения любого элемента управления
ПрименитьФильтр	Ограничивает выводимую информацию в отчетах или формах
СледующаяЗапись	Осуществляет поиск записи, следующей за текущей
НайтиЗапись	Ищет запись, удовлетворяющую задаваемому условию поиска
НаЗапись	Делает текущей запись с указанным номером, а также первую, последнюю, предыдущую, следующую
Построение пользовательского меню и выполнение команд меню	
ДобавитьМеню	Создает раскрывающееся меню
КомандаМеню	Выполняет действие одного из стандартных меню Access
Управление выводом на экран	
ВывестиНаЭкран	Выводит на экран информацию о промежуточных действиях, выполняемых при работе макроса
НаСтраницу	Переходит на заданную страницу в форме или отчете
ПесочныеЧасы	Изменяет форму курсора (указателя) на время работы макроса (например, при выполнении запроса на выборку из больших таблиц)
Развернуть	Увеличивает размер активного окна до полного размера

Окончание табл. 7.1

Макрокоманда	Назначение
Свернуть	Сворачивает активное окно в значок
СдвигРазмер	Перемещает и изменяет размер активного окна
Восстановить	Восстанавливает объект в его прежних размерах
ВыделитьОбъект	Выделяет окно с заданным объектом
Обновление	Обновляет данные элементов управления, связанных с запросом. Команда обновляет данные в таблице или форме
Установить Сообщения	Назначает (отменяет) системные сообщения или предупреждения. Не останавливает вывод на экран сообщений об ошибках
ПоказатьВсеЗаписи	Показывает все записи таблицы, игнорируя установленные ранее фильтры
ПанельИнструментов	Выводит (убирает) любую из стандартных или пользовательских панелей инструментов
Сообщения пользователю	
Сигнал	Выдает звуковой сигнал
Сообщение	Выводит предупреждающее или информационное сообщение
Работа с объектами	
КопироватьОбъект	Копирует любой объект текущей базы данных (с новым или старым именем) в другую базу данных Access
УдалитьОбъект	Удаляет объект (таблицу, запрос, форму, отчет, макрос, модуль)
ВывестиВФормате	Выводит объект в форматах приложений Windows
Переименовать	Переименовывает имя объекта в текущей БД
ОтправитьОбъект	Выводит содержание объекта в файл и передает его в электронную почту — заданному адресату
ПреобразоватьБазу Данных	Экспортирует или импортирует объекты из других БД (dBASE, Paradox, FoxPro)
ПреобразоватьТекст	Применяется при экспорте или импорте данных. Передает информацию в текстовые файлы или преобразовывает данные из текстовых файлов
Преобразовать Электронную таблицу	Аналогично — для электронных таблиц

4. Какие команды управления (кнопки) содержит панель элементов *Конструктора форм* и какова их назначение?

5. В какой последовательности необходимо разрабатывать составные формы ввода данных в таблицы, связанные отношением «один ко многим»?

6. Что означает понятие «пользовательский интерфейс»?

7. Что составляет основу метода объектно-ориентированного программирования?

8. Что такое кнопочная форма? Для каких целей разрабатывают кнопочные формы?

9. Что такое макрос и модуль и для каких задач управления базами данных их разрабатывают?

10. Какова последовательность проектирования запросов с формами для ввода условий отбора данных?

11. Какова последовательность действий при построении выражений с помощью *Конструктора*?

12. Что означают слова «Forms», «Нагрузка кафедры», «ПолеСоСписком0» в следующей записи:

[Forms]![Нагрузка кафедры]![ПолеСоСписком0]?

13. Какие кнопки имеются в форме ввода данных для просмотра записей?

14. Какова последовательность ввода данных в связанные таблицы с применением составных форм?

15. Какую последовательность действий необходимо выполнить для просмотра первой или последней записи в форме ввода данных?

16. Какую последовательность действий необходимо выполнить для ввода новой записи?

17. Какую последовательность действий необходимо выполнить для просмотра любой записи в форме ввода данных?

18. Какую последовательность действий необходимо выполнить для редактирования и последующего ввода данных в поле любой записи?

19. Для чего (для каких целей) разрабатывают отчеты в базах данных?

20. Какие способы создания отчетов имеются в Microsoft Access?

21. Для каких целей разрабатывается и что представляет собой отчет с группировкой данных?

22. Для каких целей разрабатывается и что представляет собой итоговый отчет?

Глава 8

РАЗРАБОТКА УПРАВЛЯЮЩИХ ПРОГРАММ В СРЕДЕ VISUAL BASIC FOR APPLICATIONS

8.1. Общие характеристики Visual Basic for Applications

Язык программирования Visual Basic for Applications (VBA) является общим инструментом для всех приложений Microsoft Office, позволяющим решать любые задачи программирования, начиная от автоматизации действий конкретного пользователя и кончая разработкой полномасштабных приложений, используя средства Microsoft Office как среду разработки.

Поскольку Visual Basic for Applications является объектно-ориентированным языком программирования, в данном подразделе описаны объектные модели, которые могут использоваться в Access.

Основная работа в приложениях Access — это работа с данными, поэтому мы дадим описание некоторых библиотек управления данными:

- DAO (Data Access Objects);
- ADO (ActiveX Data Objects);
- JRO (JET AND REPLICATION OBJECTS).

Рассмотрим следующие компоненты языка Visual Basic for Applications:

- процедуры и функции;
- переменные, константы и типы данных;
- область действия переменных и процедур;
- управляющие конструкции — ветвления и циклы;
- выход из циклов и процедур;
- модули.

8.2. Процедуры и функции

Основными компонентами программы на VBA являются процедуры и функции, которые представляют собой фрагменты программного кода, заключенные между операторами Sub и End Sub или между операторами Function и End Function, например:

```
Sub <имя Процедуры> (<аргумент 1>, <аргумент 2>, ...)
<оператор 1>
<оператор 2>
End Sub
```

или

```
Function <имя Функции> (<аргумент 1>, <аргумент 2>, ...)
<оператор 1>
<оператор 2>
...
<имя Функции> = <возвращаемоеЗначение>
End Function
```

Функция отличается от процедуры тем, что ее имя выступает также в качестве переменной и используется для возвращения значения в точку вызова функции.

Для того чтобы запустить на выполнение написанную процедуру или функцию, ее необходимо вызвать. Процедуру с непустым списком аргументов можно вызвать только из другой процедуры или функции. При этом ее имя со списком фактических значений аргументов необходимо задать в качестве одного из операторов VBA. Функцию можно вызвать не только с помощью отдельного оператора VBA, но и поместив ее имя со списком фактических значений аргументов прямо в формулу или выражение в программе на VBA или, например, прямо в формулу вычисляемых полях запросов, форм и отчетов Access. Процедура с пустым списком аргументов (так называемой командный макрос) может быть вызвана не только из другой процедуры или функции, но и с помощью комбинации клавиш быстрого вызова, команд раскрывающихся меню или кнопок панелей инструментов. Можно также связать такую процедуру с выполнением различных событий (например, с открытием формы или отчета, с щелчком мыши по кнопке в форме, с воздействием на элементы управления форм, в частности, на элементы управления ActiveX). Такие процедуры называют процедурами обработки событий. Понятно, что функции или процедуры, нуждающиеся в передаче им аргументов, таким способом вызвать нельзя.

Если вызываемая процедура имеет уникальное имя и находится в том же модуле, в котором находится и вызывающая процедура, то для ее вызова достаточно указать это имя и затем задать список фактических значений аргументов, не заключая его в скобки. Другой способ вызова процедуры состоит в использовании оператора Call. Сначала идет оператор Call, потом имя процедуры, а затем список параметров, в этом случае обязательно заключенный в скобки.

Функцию можно вызывать точно так же, как и процедуру, но гораздо чаще используется другой, специфический способ вызова функций — используется ее имя с заключенным в скобки списком параметров в правой части оператора присваивания.

Вот примеры вызова процедуры под именем CrossRC с передачей ей двух аргументов (константы и выражения):

```
CrossRC 7, i + 2
```

или

```
Call CrossRC (7, i + 2)
```

А вот пример вызова двух функций (Left и Mid) и использования возвращаемого ими значения в выражении:

```
yStr = Left (y, 1) & Mid (y, 2, 1)
```

Допускается два различных способа передачи переменных процедуре или функции: по ссылке и по значению. Если переменная передается *по ссылке*, то это означает, что процедуре или функции будет передан адрес в памяти для этой переменной. Вызываемая процедура может изменить значение фактического параметра. Если же фактический параметр передается *по значению*, то вызываемая процедура или функция получает только значение фактического параметра, но не саму переменную, используемую в качестве этого параметра. Все изменения полученного значения фактического параметра (если они выполняются вызываемой процедурой) не сказываются на значении переменной фактического параметра.

Способ передачи параметров процедуре или функции указывается при описании ее аргументов (формальных параметров). Имени аргумента может предшествовать явный описатель способа передачи (ByRef задает передачу по ссылке, а ByVal — по значению). Если же явное указание способа передачи параметра отсутствует, то по умолчанию подразумевается передача по ссылке.

Поясним это на примере. Пусть имеются следующие описания двух процедур — Main и Example1.

```
Sub Main ()
a = 10
b = 20
c = 30
Call Example1 (a, b, c,)
Call MsgBox (a)
Call MsgBox (b)
Call MsgBox (c)
End Sub
Sub Example1 (x, ByVal, ByRef)
x = x + 1
y = y + 1
z = z + 1
Call MsgBox (x)
Call MsgBox (y)
Call MsgBox (z)
End Sub
```

Вспомогательная процедура Example1 использует в качестве формальных аргументов переменные, описанные по-разному. Далее в теле этой процедуры каждый из них увеличивается на единицу, а затем их значения выводятся на экран с помощью функции MsgBox.

Основная процедура Main устанавливает значения переменных a, b, c, а затем передает их в качестве фактических аргументов процедуре Example1. При этом первый аргумент передается по ссылке (действует умолчание), второй — по значению, а третий — снова по ссылке.

После возврата из процедуры Example1 основная процедура также выводит на экран значения трех переменных, передавшихся в качестве аргументов.

Всего на экран выводится шесть значений: сначала числа 11, 21 и 31 (все полученные значения увеличены на 1 и выводятся процедурой Example1); затем числа 11, 20 и 31 (эти значения выводятся процедурой Main, причем переменные, переданные по ссылке, увеличились, а переменная, переданная по значению, — нет).

Программа может состоять (и обычно состоит) из многих процедур и функций, которые могут располагаться в одном или нескольких модулях. Модули группируются в проекты; при этом в одном проекте могут находиться несколько различных программ, использующих общие модули или процедуры.

Каждая из процедур, находящихся в одном модуле, должна иметь уникальное имя; при этом в проекте может содержаться несколько различных модулей. Обычно рекомендуется использовать только уникальные имена процедур в одном проекте, но допустимы и исключения. Если в проекте содержится несколько различных процедур с одним именем, то следует для уточнения имени использовать при вызове процедуры следующий синтаксис:

<имяМодуля>.<имяПроцедуры>

Если при этом имя модуля состоит из нескольких слов, то следует заключить это имя в квадратные скобки. Например, если модуль называется «Графические процедуры», а процедура — «Крестик», вызов может выглядеть следующим образом:

[Графические процедуры].Крестик

Допускается также использование процедур, расположенных и в других проектах. При этом может потребоваться еще один уровень уточнения имени

<имяПроекта>.<имяМодуля>.<имяПроцедуры>

8.3. Переменные, константы и типы данных

Как и в других языках программирования, в VBA для хранения временных значений, передачи параметров и проведения вычислений используются переменные. Кратко остановимся на основных особенностях описания и использования переменных в VBA.

Обычно, перед тем как использовать переменную, производится ее объявление, т.е. заранее указывается, какие именно имена переменных будут использованы в программе; при этом объявляется также тип данных, для хранения которых предназначена эта переменная.

В VBA, как и в обычном языке Basic, для этого используется оператор Dim со следующим синтаксисом:

Dim <имяПеременной> [As <типДанных>]

В VBA действуют следующие правила именования переменных:

- имя не может быть длиннее 255 символов;
- имя должно начинаться с буквы, за которой могут следовать буквы, цифры или символ подчеркивания;
- при написании имени не должно быть пробелов, знаков препинания или специальных символов;
- в конце имени переменной может быть добавлен еще один из шести специальных символов, указывающих тип данных: !, #, $, %, &, @. Эти символы не являются частью имени переменной. Если в программе используются одновременно имена string1$ и string1, то они ссылаются на одну и ту же строковую переменную;
- нельзя использовать одно и то же имя переменной с разными символами определения типа данных или одновременное описание типа данных и не соответствующий этому типу данных специальный символ;
- не допускается использование в качестве имен переменных ключевых слов VBA и имен стандартных объектов. Именно поэтому рекомендуется начинать имена переменных со строчной, а не с прописной буквы;
- в VBA при вводе ключевых слов и имен стандартных объектов первая буква автоматически преобразуется в прописную;
- при обозначении переменных допускается использовать в их именах буквы не только латинского алфавита, но и кириллицы.

Во многих языках программирования, например в Pascal, переменные обязательно должны быть объявлены, и эти объявления используются компилятором при резервировании памяти для переменных. В то же время в VBA, как и в его предшественнике — обычном языке Basic, допускается использование необъявленных переменных.

Одним из самых опасных источников трудно обнаруживаемых ошибок в языках программирования, допускающих применение неописанных переменных, служат опечатки в написании имен переменных. Такие опечатки истолковываются транслятором как появление еще одной, новой переменной, отличной от ранее использовавшейся, и не воспринимаются как ошибки.

В VBA принято поистине «соломоново решение» — разрешение этой проблемы предоставлено самому программисту. Для этого имеется оператор Option Explicit. Описание модуля должно начинаться с этого оператора. В этом случае VBA будет требовать обязательного объявления переменных в этом модуле и генерировать сообщения об ошибке всякий раз, когда встретится необъявленная переменная.

В табл. 8.1 приведен перечень используемых типов данных VBA.

При описании переменной указание типа данных может быть опущено. Тип переменной может в таком случае определяться последним символом имени переменной: @, #, %, &, !, $ (соответственно Currency, Double, Integer, Long, Single или String). Например, поскольку символ «$» является символом определения типа для строковых данных, то переменная под именем text$ ав-

Таблица 8.1

Типы данных VBA

Тип данных	Описание
ARRAY	Массив переменных. Для ссылки на конкретный элемент массива используется индекс. Требуемая память зависит от размеров массива
BOOLEAN	Принимает одно из двух логических значений: *True* (Истина) или *False* (Ложь). Требуемая память — 2 байт
BYTE	Число без знака — от 0 до 255. Требуемая память — 1 байт
CURRENCY	Используется для произведения денежных вычислений с фиксированным числом знаков после десятичной запятой в тех случаях, когда важно избежать возможных ошибок округления. Диапазон возможных значений: –922 337 203 685 477,5808 до 922 337 203 685 477,5807. Требуемая память — 8 байт. Для определения типа по умолчанию используется символ «@»
DATE	Используется для хранения дат. Диапазон возможных значений: от 1 января 0100 г. до 31 декабря 9999 г. Требуемая память — 8 байт

Тип данных	Описание
DOUBLE	Числовые значения с плавающей точкой двойной точности. Диапазон возможных значений для отрицательных чисел: от –1,79769313486232E308 до –4,94065645841247E–324. Диапазон возможных значений для положительных чисел: от 4,94065645841247E–24 до 1,79769313486232E308. Требуемая память — 8 байт. Для определения типа по умолчанию используется символ «#»
INTEGER	Короткие целые числовые значения. Диапазон возможных значений: от –32 768 до 32 767. Требуемая память — 2 байт. Для определения типа по умолчанию используется символ «%»
LONG	Длинные целые числовые значения. Диапазон возможных значений: от –2 147 483 648 до 2 147 483 647. Требуемая память — 4 байт. Для определения типа по умолчанию используется символ «&»
OBJECT	Используется только для хранения ссылок на объекты. Требуемая память — 4 байт
SINGLE	Числовые значения с плавающей точкой обычной точности. Диапазон возможных значений для отрицательных чисел: от –3,402823E38 до –1,401298E–45. Диапазон возможных значений для отрицательных чисел: от 1,401298E–45 до 3,402823E38. Требуемая память — 4 байт. Для определения типа по умолчанию используется символ «!»
STRING	Используется для хранения строковых значений. Длина строки — от 0 до 64 Кбайт. Требуемая память — 1 байт на символ. Для определения типа по умолчанию используется символ «$»
VARIANT	Может использоваться для хранения различных типов данных. Требуемая память — 16 байт плюс 1 байт на каждый символ строковых значений. Символ определения типа по умолчанию отсутствует
ОПРЕДЕЛЯЕМЫЙ ПОЛЬЗОВАТЕЛЕМ	Определяемые пользователем типы данных. Назначение и размер выделяемой памяти зависит от определения типа. Используется для описания структур данных. Позволяет хранить множество различных значений различных типов данных

томатически становится переменной типа «строка символов». Если же последний символ не является ни одним из перечисленных выше символов и явное указание типа тоже не используется, то в этом случае переменной будет назначен по умолчанию тип данных Variant, который позволяет хранить в ней данные любого типа.

Нельзя использовать в одной и той же процедуре переменные, отличающиеся друг от друга только специальным символом определения типа в конце переменной. Например, не допускается одновременное использование переменных var$ и var%. Не допускается и явное объявление переменной, уже содержащей символ определения типа в конце имени, с помощью описателя «AS <типПеременной>» (даже если такое определение не противоречит обычному применению символа определения типа). Например, вы получите сообщение об ошибке, попытавшись ввести любое из следующих определений:

```
Dim var1$ As String
Dim var2% As Integer
```

Для определения типа данных аргументов процедуры или функции используется описание типа данных непосредственно в заглавной строке процедуры или функции. Например, следующая заглавная строка процедуры описывает ее параметры как переменные строкового типа:

```
Sub SplitStr(str1 As String, str2 As String, str3 As String).
```

Определение типа данных возвращаемого функцией значения завершает заглавную строку функции, например,

```
Function FindSplitSpace (str1 As String) As Integer
```

описывает возвращаемое функцией значение как переменную короткого типа.

Рассмотрим использование именованных констант. Для их описания используется оператор Const, схожий с оператором описания переменных Dim. Синтаксис этого оператора:

```
Const <имяКонстанты> (As <типДанных>] = <выражение>
```

где <выражение> — это любое значение или формула, возвращающая значение, которое должно использоваться в качестве константы.

Например, следующий оператор определяет целочисленную константу maxLen:

```
Const maxLen% = 30
```

Кроме описываемых пользователем констант существуют еще предопределенные встроенные константы, которые используют в тексте программ без предварительного описания.

При именовании встроенных констант используется стандартное соглашение, позволяющее определить, к объектам какого приложения относится эта константа. Например, имена встроенных констант, относящихся к объектам Access, начинаются с префикса «ac»; относящихся к объектам Excel — с префикса «xl»; относящихся к объектам Word — с префикса «wd»; относящихся к объектам VBA — с префикса «vb».

Например, в команде

```
DoCmd. OpenForm «Orders», acNormal, stLinkCriteria
```

используется встроенная константа Access acNormal.

Ссылки на объекты. Кроме обычных переменных в Visual Basic часто используются переменные, представляющие собой ссылку на объект. Использование переменных для ссылок на объекты позволяет не только сократить и упростить текст программы, но и существенно ускорить ее работу.

Использование переменной-объекта несколько отличается от использования обычных переменных. В этом случае нужно не только объявить такую переменную, но и перед использованием назначить ей соответствующий объект с помощью специального оператора Set.

Синтаксис такого объявления и назначения:

```
Dim <имяПеременной> As Object
Set <имяПеременной> = <ссылкаНаОбъект>
```

Иногда при объявлении такой переменной удобно заранее указать конкретный тип объекта. При этом можно использовать любой конкретный объект из объектной модели Office. Например:

```
DIM MYBASE AS DATABASE
Set MyBase = DBEngine.Workspaces(0).Databases(0)
```

После такого объявления и назначения можно использовать переменную MyBase для обращения к текушей открытой базе данных. Такая ссылка быстрее обрабатывается, и программа, использующая переменные для прямых ссылок на объекты вместо сложных иерархических ссылок, использующих при этом большое число операторов уточнения (точек), работает быстрее.

Массивы. Массив — это переменная, в которой хранится одновременно несколько значений одинакового типа. Формальное определение массива — это совокупность однотипных индексированных переменных.

Число используемых индексов массива также может быть различным. Чаще всего используются массивы с одним или двумя индексами. В VBA допускается использовать до 60 индексов. О числе индексов массива говорят как о размерности массива. Массивы с одним индексом называют одномерными, с двумя — двумерными и т.д.

Прежде чем использовать массив, нужно обязательно объявить его с помощью оператора Dim и указать тип хранящихся в массиве значений. Все значения в массиве обязаны принадлежать к одному типу данных. Это ограничение можно обойти, используя при объявлении массива тип Variant. В этом случае элементы массива смогут принимать значения различных типов данных.

Синтаксис оператора объявления массива:

Dim <имяМассива> (<размер1>, <размер2>, ...) As <типДанных>

Указанные в скобках величины задают размеры массива — число индексов и максимально допустимое значение каждого конкретного индекса. При этом индексирование элементов массива по умолчанию начинается с нуля. Так, объявление

DIM ARRAY (9) AS INTEGER

определяет одномерный массив из 10 элементов, являющихся переменными целого типа, а объявление

DIM ARRAY (4, 9) AS VARIANT

определяет двумерный массив из пятидесяти (5×10) элементов, являющихся переменными универсального типа Variant.

При объявлении массива можно указать не только верхнюю границу индекса, но и его нижнюю границу, т.е. явно задать диапазон изменения конкретного индекса массива, причем нижняя граница может быть любым целым числом. Синтаксис такого определения:

Dim <имяМассива> (<мин1> To <макс1>,) As <типДанных>

В приведенных примерах говорилось о массивах фиксированного размера, число элементов в которых явно указано в операторе Dim при описании массива. Такие массивы называются статическими. В VBA допускается использование и динамических массивов, размеры которых при описании не фиксируются. Определение размера динамического массива может быть сделано непосредственно во время выполнения программы.

При определении динамического массива в операторе Dim после имени массива должны стоять пустые скобки и описание типа переменных. Число индексов и диапазон их изменения не задаются. Перед тем как использовать массив, необходимо ввестн оператор ReDim, который задаст размерность и диапазоны изменения индексов динамического массива.

Синтаксис объявления и определения размеров динамического массива:

Dim <имяМассива> () As <типДанных>
ReDim <имяМассива> (<размер1>, < размер2>, ...)

Приведем пример объявления и определения размеров динамического массива, а затем последующее изменение размерности этого массива:

```
Dim dArray () As Variant
ReDim dArray (1, 2)
dArray (0, 0) = 2
dArray (0, 1) = 3
k = dArray (0, 0) + dArray (0, 1)
ReDim dArray (k)
dArray (0) = (Строка1)
```

В этом примере массив dArray сначала определяется как двумерный массив из шести элементов, а затем переопределяется как одномерный массив, причем верхняя граница индекса задается значением переменной k.

8.4. Область действия переменных и процедур

Все процедуры, функции, переменные и константы в VBA имеют свою область действия. Это означает, что все они могут использоваться только в определенном месте программного кода — там, где они описаны. Например, если переменная A описана с помощью оператора Dim в теле процедуры с именем Proc1, то именно эта процедура и является ее областью действия. Если имеется другая процедура Proc2, то в ней нельзя использовать эту же переменную без объявления.

Области действия переменных. То, в каком месте программы и как именно описана переменная, и то, как долго она «живет» в памяти и сохраняет присвоенное ей значение, определяет ее область действия. Имеется три различных уровня при определении области действия переменных: уровень процедуры, уровень модуля и уровень проекта.

Чтобы определить переменную на *уровне процедуры*, ее описание помещается в тело этой процедуры.

Чтобы определить процедуру на *уровне модуля* и сделать ее тем самым доступной для совместного использования во всех процедурах этого модуля, следует поместить ее описание в секции объявлений модуля — перед текстом каких-либо процедур или функций. При этом может использоваться и явное описание области действия. В этом случае вместо ключевого слова «Dim» используется ключевое слово «Private».

Чтобы описать переменную на *уровне проекта*, необходимо расположить ее описание в секции объявлений одного из модулей проекта; при этом обязательно должно использоваться ключевое слово «Public».

Описанные таким образом переменные могут использоваться в любом модуле проекта.

Все сказанное выше относится к описанию и определению области действий констант и массивов.

Для переменных имеется еще один способ их описания, не изменяющий их уровня, но позволяющий сохранить значение переменной, описанной на уровне процедуры и после завершения ее работы.

Для этого следует использовать переменную Static, определяя ее как статическую переменную. Такая переменная сохраняет выделенное ей место в памяти и свое значение даже после завершения процедуры, в которой она была описана и использована.

Тем не менее статическая переменная не может быть использована в других процедурах. Изменяется лишь время ее жизни, но не область действия. Если произойдет повторный вызов той же самой процедуры, в которой была описана статическая переменная, то эта переменная сохранит свое прежнее значение, которое она имела в момент завершения работы процедуры при предыдущем вызове. Обыкновенные (не статические) переменные всякий раз инициализируются заново и получают при входе в процедуру пустые значения.

Области действия процедур и функций. Процедуры и функции имеют только два уровня областей действия: уровень модуля и уровень проекта. По умолчанию используется уровень проекта. В проекте процедура или функция может быть вызвана любой другой процедурой или функцией. При описании процедур и функций на уровне проекта может также использоваться необязательное ключевое слово «Public». Наличие или отсутствие этого слова не оказывает на процедуру никакого воздействия.

Если необходимо описать процедуру, используемую только на уровне модуля, то для этого применяется ключевое слово «Private». Однако такое описание процедуры сужает область действия для процедуры и запрещает ее использование как самостоятельной. Ее можно вызвать только из другой процедуры.

При описании процедур или функций может использоваться и ключевое слово «Static». Оно никак не влияет на область действия процедуры, но воздействует на все переменные, описанные внутри этой процедуры или функции. В этом случае все локальные переменные получают статус Static и тем самым сохраняются в памяти после завершения процедуры; при повторном ее вызове они сохраняют свои прежние значения.

Рассмотрим следующий пример.

```
Public A1 As String
Private A2 As Integer
Dim A3 As Single
Sub Proc1()
```

```
    Dim A4 As Integer
    Static A5 As Integer
    A1 = "Текстовая строка"
    A2 = 2
    A3 = 3.14
    A4 = A4 + 4
    A5 =A5 + 5
    MsgBox A4
    MsgBox A5
End Sub
Sub Proc2()
    Proc1
    MsgBox A1
    MsgBox A2
    MsgBox A3
    MsgBox A4
    MsgBox A5
    Proc1
End Sub
```

В этом примере переменная A1 определена на уровне всего проекта (использовано ключевое слово «Public»), переменные A2 и A3 определены на уровне модуля, переменная A4 — только на уровне процедуры Proc1, переменная A5, хотя и определена в теле процедуры Proc1, но описана как статическая переменная.

При вызове процедуры Proc2 произойдет следующее: из этой процедуры будет вызвана процедура Proc1, которая присвоит значения всем пяти переменным A1, A2, A3, A4 и A5, а затем выведет текущие значения переменных A4 и A5 в диалоговом окне (*MsgBox*).

После завершения этой процедуры будут выведены текущие значения всех пяти переменных из процедуры Proc2. При этом произойдет следующее. Переменные A1...A3 сохранят свои значения, поскольку они описаны на уровне модуля, а переменные A4 и A5 примут значения 0 (пустые значения), поскольку областью действия этих переменных является только процедура Proc1.

Затем происходит еще один вызов процедуры Proc1 и она вновь начинает изменять и выводить на экран значения переменных A4 и A5. При этом переменная A4 вновь получит значение 4, поскольку при новом вызове процедуры для этой переменной будет заново выделена память и она будет инициализирована пустым значением. В отличие от A4 переменная A5, описанная как статическая переменная, сохранит свое прежнее значение от предыдущего вызова этой процедуры и в результате ее значение при повторном вызове окажется равным 10.

8.5. Управляющие конструкции — ветвления и циклы

Как во всех языках программирования, в VBA имеются различные управляющие конструкции, позволяющие изменять порядок выполнения программы. Если не использовать управляющие конструкции, то происходит последовательное выполнение операторов языка программирования, начиная с самого первого и кончая последним. Хотя в некоторых самых простых случаях этого бывает достаточно, иногда все-таки требуется изменять порядок выполнения операторов при выполнении определенных условий, либо пропуская выполнение некоторых операторов, либо, наоборот, многократно повторяя их.

Практика показывает, что для реализации любых алгоритмов обработки информации достаточно иметь два вида инструкций управления: ветвления и циклы.

Ветвления. Управляющие конструкции ветвления позволяют проверить некоторое условие, а затем, в зависимости от результатов этой проверки, выполнить ту или иную группу операторов. Для организации ветвлений в VBA используются различные формы оператора ветвления If и оператор выбора Select Case.

Простейшая краткая форма оператора If используется для проверки одного условия, а затем, в зависимости от результата проверки, либо для выполнения, либо пропуска одного оператора или блока из нескольких операторов. Краткая форма оператора ветвления If может иметь как однострочную, так и блочную форму. В одну строку краткая форма If может быть записана так:

```
If <условие> Then <оператор>
```

В блочной форме краткое ветвление выглядит следующим образом:

```
If <условие> Then
<оператор1>
.....
<операторN>
End If
```

В качестве условия можно использовать логическое выражение, возвращающее значение *True* (Истина) или *False* (Ложь), или любое арифметическое выражение. Если используется арифметическое выражение, то нулевое значение этого выражения эквивалентно логическому значению *False*, а любое ненулевое значение эквивалентно значению *True*. В том случае когда условие возвращает значение *False*, оператор или блок операторов, составляющих тело краткого оператора ветвления и заключенных ключевыми словами «Then» и «End If», не будет выполняться.

Примечание. При записи краткого оператора ветвления в одну строку ключевые слова «End If» не используются.

Полная форма оператора If используется в тех случаях, когда имеются два различных блока операторов и по результатам проверки условия нужно выполнить один из них. Такая форма If не может записываться в одну строку и всегда имеет блочную форму записи:

```
If <условие> Then
<блокОператоров1>
ELSE
<блокОператоров2>
End If
```

Если условие истинно, то выполняется первый блок операторов, заключенных между ключевыми словами «Then» и «Else»; в противном случае — второй блок, заключенный между ключевыми словами «Else» и «End If».

Иногда приходится делать выбор одного действия из целой группы альтернативных действий на основе проверки нескольких различных условий. Для этого можно использовать цепочку операторов ветвления If...Then...Eise If:

```
If <условие1> Then
<блокОператоров1>
Else If <условие2> Then
<блокОператоров2>
Else If <условие3> Then
<блокОператоров3>
.....
Else If <условиеN> Then
<блокОператоровN>
Else
<блокОператоров_Else>
End If
```

Такие цепочки операторов If...Then...Else If обладают большой гибкостью и позволяют решить многие проблемы, однако если выбор одной из нескольких возможностей все время основан на различных значениях одного и того же выражения, гораздо удобнее использовать специально предназначенный для этого оператор выбора Select Case, имеющий следующий синтаксис:

```
Select Case <проверяемоеВыражение>
Case <списокЗначений1>
<блокОператоров 1>
Case <списокЗначений2>
<блокОператоров2>
```

```
Case <списокЗначений3>
<блокОператоров3>
.....
Case Else
<блокОператоров_Else>
End Select
```

Проверяемое выражение вычисляется в начале работы оператора Select Case. Это выражение может возвращать значение любого типа.

Список выражений представляет собой одно или несколько выражений, разделенных запятой. При выполнении оператора проверяется, соответствует ли хотя бы один из элементов этого списка проверяемому выражению. Эти элементы списка выражений могут иметь одну из следующих форм:

<выражение>

(в этом случае проверяется, совпадает ли значение проверяемого выражения с заданным выражением);

<выражение1> To <выражение2>

(в этом случае проверяется, находится ли значение проверяемого выражения в указанном диапазоне значений);

If <логическийОператор> <выражение>

(в данном случае проверяемое выражение сравнивается с указанным значением с помощью логического оператора; например, условие If >= 20 считается выполненным, если проверяемое значение не меньше 20).

Если хотя бы один из элементов списка соответствует проверяемому выражению, то выполняется соответствующая группа операторов и на этом выполнение оператора Select Case заканчивается, а остальные списки выражений не проверяются. Если же ни один из элементов всех этих списков не соответствует значению проверяемого выражения, то выполняются операторы группы Else, если они имеются.

Циклы. В VBA имеется достаточный набор средств организации циклов, которые можно разделить на две группы:

- циклы с условием Do...Loop;
- циклы с перечислением For...Next.

Циклы типа Do...Loop используются в тех случаях, когда заранее не известно, сколько раз должно быть повторено выполнение блока операторов, составляющего тело цикла. Такой цикл продолжает свою работу до тех пор, пока не будет выполнено определенное условие. Существуют четыре вида циклов Do...Loop которые различаются типом проверяемого условия и временем выполнения этой проверки. Синтаксис этих четырех конструкций циклов приведен в табл. 8.2.

Таблица 8.2

Синтаксис конструкций цикла

Конструкция	Описание
Do While <условие> <блокОператоров> Loop	В данной конструкции значение условия выполнения цикла проверяется до выполнения операторов
Do<блокОператоров> Loop While <условие>	В данной конструкции значение условия выполнения цикла проверяется после выполнения операторов
Do Until <условие> <блокОператоров> Loop	В данной конструкции указывается значение условия прекращения работы цикла, которое задается до выполнения операторов
Do <блокОператоров> Loop Until <условие>	В данной конструкции указывается значение условия прекращения работы цикла, которое задается после выполнения операторов

Имеется также две разновидности оператора цикла с перечислением For...Next. Очень часто при обработке массивов, а также в тех случаях, когда требуется повторить выполнение некоторой группы операторов заданное число раз, используется цикл For...Next со счетчиком. В отличие от циклов Do...Loop данный тип цикла использует специальную переменную, называемую счетчиком, значение которой увеличивается или уменьшается при каждом выполнении тела цикла на заданную величину. Когда значение этой переменной достигает заданного значения, выполнение цикла заканчивается. Синтаксис этого вида цикла (в квадратных скобках заключены необязательные элементы):

```
For <счетчик> = <начальноеЗначение> To <конечноеЗначение>
   [Step <приращение>]
   <блокОператоров>
Next <счетчик>
```

Приведем несколько замечаний к приведенному: фрагменту конструкции цикла:

- <приращение> может быть как положительным, так и отрицательным числом. Если использовать отрицательное приращение, то для того, чтобы тело цикла выполнилось хотя бы один раз, конечное значение должно быть меньше либо равно начальному значению;
- после завершения работы цикла For...Next переменная, которая использовалась в качестве счетчика, получает значение, обязательно большее конечного значения, если приращение положительно, и меньшее конечного значения, если приращение отрицательно;

• если начальное и конечное значения совпадают, то тело цикла выполняется лишь один раз.

Рассмотрим еще одну разновидность цикла For...Next, часто использующуюся в VBA при обработке объектов, составляющих массив или семейство однородных объектов.

В этой разновидности цикла For Each...Next счетчик отсутствует, а тело цикла выполняется для каждого элемента массива или семейства объектов. Синтаксис такого цикла:

```
For Each <элемент> In <совокупность>
<блокОператоров>
Next [<элемент>]
```

где <элемент> — это переменная, используемая для ссылки на элементы семейства объектов; <совокупность> — это имя массива или семейства.

Приведем пример использования подобного цикла. Следующая процедура предназначена для выдачи на печать списка всех полей для всех таблиц текущей открытой базы данных:

```
Public Sub EnumerateAllFields ()
Dim MyBase As Database
Dim tdf As TableDef, fid As Field
Set MyBase = DBEngine.Workspaces(0).Databases(0)
For Each tdf In MyBase.TableDefs
Debug.Print "Таблица: " & tdf.Name
For Each fid In tdf. Fields
Debug. Print "Поле: " & fid. Name
Next fid
NEXT TDF
Set MyBase = Nothing
End Sub
```

В данном примере операторами Dim были объявлены следующие переменные:

- MyBase — объект «База данных»;
- tdf — таблица базы данных;
- fid — поле таблицы.

Оператор Set назначает переменной MyBase значение имени текущей, открытой базы данных. Далее для каждого определения таблицы выполняется вывод на печать названия таблицы, а затем вложенный цикл такого же типа печатает названия всех ее полей.

8.6. Выход из циклов и процедур

Обычно выполнение процедуры заканчивается после ее последнего оператора, а выполнение цикла — после достижения условия завершения его работы.

Однако часто бывает нужно прекратить выполнение процедуры или цикла досрочно, не выполняя некоторых операторов процедуры или повторений цикла.

Например, если при выполнении процедуры произошла ошибка, которая делает продолжение работы бессмысленным, то можно выполнить команду немедленного выхода из процедуры.

Другой пример: если цикл For...Next используется для поиска нужного значения в массиве, то после того, как нужный элемент массива найден, нет смысла продолжать дальнейший просмотр элементов массива. Досрочный выход из управляющей конструкции можно осуществить с помощью одного из операторов Exit. Для досрочного выхода из циклов Do...Loop используется оператор Exit Do, а для выхода из циклов For — оператор Exit For. Для досрочного выхода из процедур и функций используются соответствующие операторы Exit sub и Exit Function.

Необходимо отметить, что хотя использование оператора Exit может быть вполне оправданным, применять его следует в крайних случаях. Излишне частое применение этого оператора затрудняет понимание написанного текста и его отладку.

Рассмотрим следующие примеры:

```
ub = Ubound (dArrey)
fFound = False
For I =LBound (dArrey) to ub
   If dArrey(i) = searchValue Then
fFound = True
Exit For
NEXT
```

Использование оператора Exit вполне можно избежать. Это показано на следующем примере:

```
i = LBound (dArrey)
ub = Ubound (dArrey)
fFound = False
Do
   If dArrey(i) = searchValue Then fFound = True
   i = i + 1
Loop Until (i > ub) or fFound
```

8.7. Модули

Модуль — это набор описаний и процедур на языке Visual Basic для приложений, собранных в одну программную единицу. Существует два основных типа модулей: модули класса и стандартные модули. Каждая процедура в модуле может быть либо процедурой-функцией Function, либо процедурой-подпрограммой Sub.

Модули класса. Модулями класса являются программы, связанные с определенной формой или отчетом. Модули класса содержат процедуры обработки событий, запускаемые в ответ на событие в форме или отчете. Процедуры обработки событий используются для управления поведением формы или отчета и их откликом на события, такие как, например, нажатие кнопки.

При создании первой процедуры обработки события для формы или отчета автоматически создается связанный с ней модуль формы или отчета. Для просмотра модуля для формы или отчета достаточно нажать кнопку *Программа* на панели инструментов в режиме *Конструктор*.

В процедурах модулей форм и отчетов могут содержаться вызовы процедур, добавленных в стандартные модули.

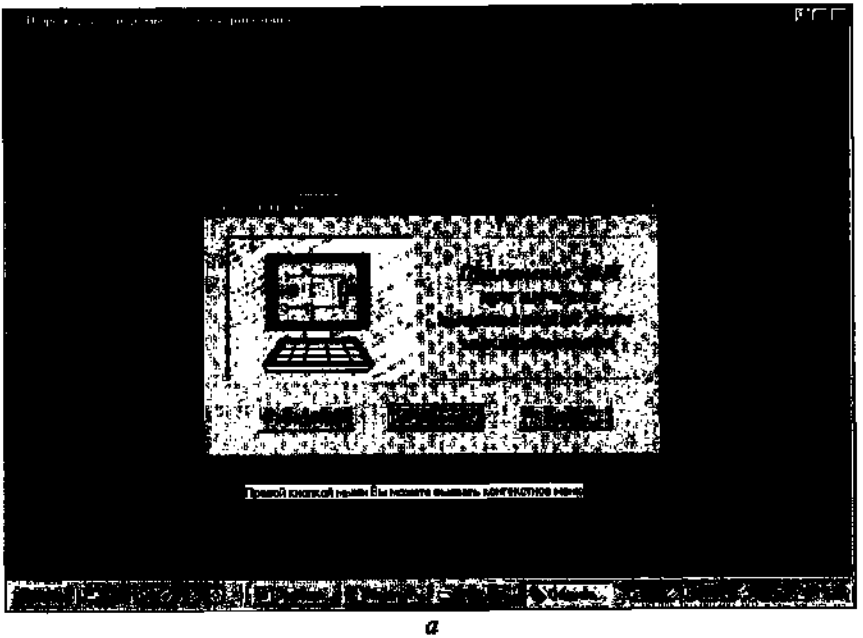
Стандартные модули. Стандартные модули содержат общие процедуры, не связанные ни с каким объектом, а также процедуры, которые могут быть запущены из любого окна базы данных.

Стандартные модули могут разрабатываться для объявления глобальных переменных. Если в процедурах модуля не указаны ссылки на конкретные объекты приложения (формы, отчеты, элементы управления), то такой модуль может применяться в других приложениях Access.

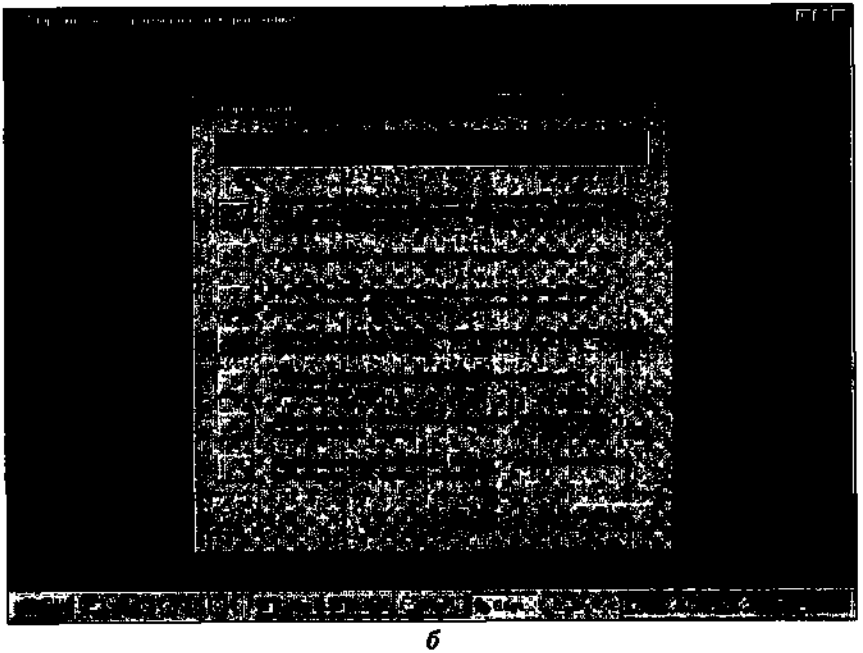
На рис. 8.1, *а* показана главная форма, а на рис. 8.1, *б* — кнопочная форма режимов работы с обучающей программы в электронной версии учебного пособия по теоретическим основам электротехники.

Ниже представлен текст модуля, управляющего работой с этими и другими формами учебного пособия, показанными на рис. 8.1.

```
'Данный модуль хранит различные процедуры общего назначения
'Разработчики: Фуфаев Э.В. и Фуфаева Л.И.
Option Explicit
'Ниже приведены переменные, которые отвечают за передачу параметров состояния
'между формами.
Public ShowMax As Boolean
Private Default As Boolean
Private FrmBackColor As Long
'Процедура осуществляет инициализацию всех форм в программе, позиционирование их в центре экрана.
Public Sub InitForm(TForm As Form, TPicture As PictureBox, TFrame As Frame, TButton As CommandButton, TSSubItem As Menu)
TForm.Left = 0
TForm.Top = 0
TForm.Width = Screen.Width '12000
TForm.Height = Screen.Height ' – 430 '8570
```



а



б

Рис. 8.1. Формы:

а — главная; *б* — кнопочная форма выбора режимов работы с обучающей программой

```
    TPicture.Left = 0
    TPicture.Top = 0
    TPicture.Width = Screen.Width ' - 15 * 6
    TPicture.Height = Screen.Height ' - 15 * 25
    TPicture.BackColor = FrmBackColor
    If ShowMax Then TForm.WindowState = 0
    If Not ShowMax Then TForm.WindowState = 2
    If Default Then Lines TPicture
    TButton.Left = (TPicture.Width - TFrame.Width * 15)/30
    TButton.Top = (TPicture.Height - TFrame.Height * 15 - 750)/30
    TFrame.Left = TButton.Left + 2
    TFrame.Top = TButton.Top + 2
    If Default Then TSSubItem.Checked = True
    If Not Default Then TSSubItem.Checked = False
    End Sub
    'Процедура прорисовки фона главной формы с использованием перехода цвета.
    Public Sub Lines(Picture As PictureBox)
    Dim i, J, Stp, col As Integer
    J = 0
    col = 255
    Stp = Step
    For i = 0 To (Picture.Height\15)
    J = J + 1
    If (J = Stp) And (col <> 1) Then
    col = col - 1
    J = 0
    End If
    Picture.Line (Picture.Left, i)-(Picture.Left + Picture.Width, i),
RGB(0, 0, col)
    Next
    End Sub
    'Вспомогательная процедура, рассчитывающая шаг изменения цвета фона
    Public Function Step() As Integer
    Dim i, Col_ As Integer
    i = 0
    Col_ = 0
    Do Until Col_ > (Screen.Height\15)
    Col_ = Col_ + 255
    i = i + 1
    Loop
    If Col_ - (Screen.Height \ 15) > 100 Then i = i - 1
    Step = i
    End Function
    'Считывание параметров программы из реестра
```

```
    Public Sub GetFromRegister()
    Dim Ret As Long
    Ret = GetSetting ("Book", "Init", "BackColor", vbApplicationWorkspace)
"7303023)
    FrmBackColor = Ret
    Ret = GetSetting("Book", "Init", "Default", 1)
    If Ret = "0" Then Default = False
    If Ret = "1" Then Default = True
    Ret = GetSetting("Book", "Init", "Size", 2)
    If Ret = "0" Then ShowMax = True
    If Ret = "2" Then ShowMax = False
    End Sub
    'Сохранение параметров программы в реестр
    Public Sub SavetoRegister()
    SaveSetting "Book", "Init", "BackColor", FrmBackColor
    If Default Then
    SaveSetting "Book", "Init", "Default", "1"
    Else
    SaveSetting "Book", "Init", "Default", "0"
    End If
    If ShowMax Then
    SaveSetting "Book", "Init", "Size", "0"
    Else
    SaveSetting "Book", "Init", "Size", "2"
    End If
    End Sub
    'Обработка выбора первого пункта контекстного меню главной
формы
    Public Sub Item1Clk(ColorDialog As CommonDialog, TPicture As
PictureBox, TItem As Menu)
    On Error GoTo ErrHandler
    ColorDialog.Flags = cdlCCRGBInit
    ColorDialog.ShowColor
    FrmBackColor = ColorDialog.Color
    TPicture.BackColor = FrmBackColor
    TItem.Checked = False
    Default = False
    Exit Sub
    ErrHandler: Exit Sub
    End Sub
    'Обработка выбора второго пункта контекстного меню главной
формы
    Public Sub Item2Clk(TPicture As PictureBox, TItem As Menu)
    If TItem.Checked Then
    Default = False
    TItem.Checked = False
```

```
TPicture.BackColor = TPicture.BackColor
Else
Default = True
TItem.Checked = True
Lines TPicture
End If
End Sub
```

Итак, мы познакомились с основными понятиями языка программирования VBA.

Рассмотрим некоторые способы его применения для автоматизации работы с данными в системе Access.

На рис. 8.2 представлена форма для составления классификатора технологических процессов изготавливаемых на предприятии изделий.

Классификатор предназначен для быстрого поиска обозначения сборочной единицы (№ сборочного чертежа).

Данный классификатор представляет собой четырехуровневую иерархическую систему, которая может быть представлена в виде четырех таблиц (списков), связанных отношениями «один ко многим»

- первая таблица содержит наименования типов изделий;

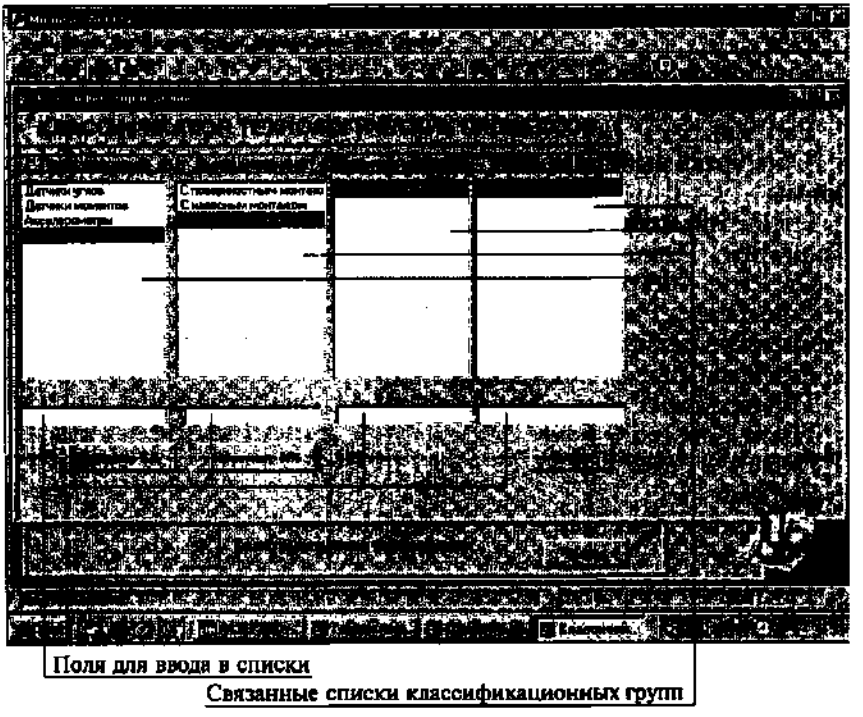


Рис. 8.2. Форма *Классификатор технологических процессов*

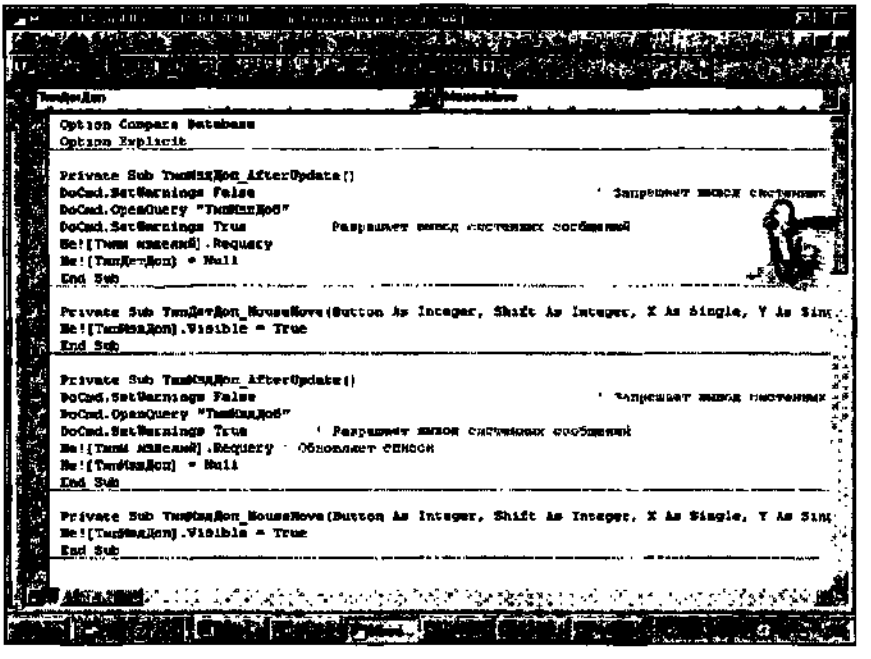


Рис. 8.3. Окно *Построитель программ* с фрагментом программы

• вторая таблица содержит наименования видов изделий для каждого типа;

• третья таблица содержит наименования изделий, в данном примере — сборочных единиц, входящих в конкретный вид;

• четвертая таблица содержит обозначение (номер) чертежа сборочной единицы.

Алгоритм заполнения классификатора состоит из следующих действий:

• поместить курсор в поле ввода данных первого списка (типы изделий);

• ввести наименование типа;

• выделить тип изделия и переместить курсор в поле ввода данных второго списка — *Виды изделий*;

• повторить указанные действия для заполнения следующих списков.

Для выполнения этих процедур рассмотрим программу на языке VISUAL BASIC.

Разработка программ производится по аналогии с макросами и выражениями в окне *Построитель программ*.

Окно *Построитель программ* с фрагментом текста программы приведено на рис. 8.3. Последовательность разработки программы:

• открыть форму в режиме *Конструктор*;

• выделить объект (поле);
• вызвать окно *Свойства объекта.*

В окне *Свойства* в строке соответствующего свойства вызвать *Построитель программ.* В результате появится окно *Построитель программ* с начальными операторами языка:

```
Private Sub — инструкция, объявляющая новую подпрограмму.
(Текст программы)
.....
End Sub — инструкция, закрывающая подпрограмму.
```

Ниже приведены тексты подпрограмм выполняемых действий при заполнении формы *Классификатор технологических процессов* (см. рис. 8.2).

```
Option Compare Database
Option Explicit
Private Sub ТипИздДоп_AfterUpdate() ' Начало программы дополнения списка Типы изделий
DoCmd.SetWarnings False ' Запрещает вывод системных сообщений
DoCmd.OpenQuery "ТипИздДоб"
DoCmd.SetWarnings True ' Разрешает вывод системных сообщений
Me![Типы изделий].Requery
Me![ТипИздДоп] = Null
End Sub

Private Sub ТипИздДоп_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
Me![ТипИздДоп].Visible = True
End Sub

Private Sub ТипИздДоп_AfterUpdate()
DoCmd.SetWarnings False ' Запрещает вывод системных сообщений
DoCmd.OpenQuery "ТипИздДоб"
DoCmd.SetWarnings True ' Разрешает вывод системных сообщений
Me![Типы изделий].Requery ' Обновляет список
Me![ТипИздДоп] = Null
End Sub

Private Sub ТипИздДоп_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
Me![ТипИздДоп].Visible = True
End Sub

Private Sub ВидИздДоп_AfterUpdate()
```

```
    DoCmd.SetWarnings False ' Запрещает вывод системных сообщений
    DoCmd.OpenQuery "ВидИздДоб"
    DoCmd.SetWarnings True ' Разрешает вывод системных сообщений
    Me![Виды изделий].Requery ' Обновляет список
    Me![ВидИздДоп] = Null
    End Sub
    Private Sub МидИздДоп_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Me![ВидИздДоп].Visible  = True
    End Sub

    Private Sub НаименИзд_AfterUpdate()
    Me![НаименИзд].Requery
    End Sub

    Private Sub НаименИзд_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Me![НаименИзд].SetFocus
    End Sub

    Private Sub Типы_изделий_AfterUpdate()
    Me![Виды изделий].Requery
    Me![Наимен изделий].Requery
    Me![Обозн изделий].Requery
    End Sub

    Private Sub Типы изделий_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Me![Типы изделий].SetFocus
    End Sub

    Private Sub Обозн изделий MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Me![Обозн изделий].SetFocus
    End Sub

    Private Sub ВидИздДоп_AfterUpdate()
    DoCmd.SetWarnings False ' Запрещает вывод системных сообщений
    DoCmd.OpenQuery "ВидИздДоб"
    DoCmd.SetWarnings True ' Разрешает вывод системных сообщений
    Me![Виды изделий].Requery ' Обновляет список
    Me![ВидИздДоп] = Null
    End Sub

    Private Sub ВидИздДоп_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
Me![ВидИздДоп].Visible = True
End Sub

Private Sub Виды изделийAfterUpdate()
Me![Наимен изделий].Requery
Me![Обозн изделий].Requery
End Sub

Private Sub Виды изделий MouseMove(Button As Integer, Shift
As Integer, X As Single, Y As Single)
Me![Виды изделий].SetFocus
End Sub
```

Контрольные вопросы

1. На какие классы разделяются макросы?
2. Перечислите способы создания макросов, предназначенных для выполнения действий при «нажатии» кнопки на форме.
3. Какова последовательность действий при создании макросов с использованием конструктора макросов?
4. Сформулируйте последовательность действий при создании специальной панели инструментов для открытой базы данных.
5. Какая последовательность действий необходима при создании специального контекстного меню для активной базы данных?
6. Для чего служат кнопочные формы и формы-заставки?
7. В каких случаях разрабатывают модули — подпрограммы, написанные на языке VISUAL BASIC?
8. Используя справочную систему Access, самостоятельно разберите назначение выражений в приведенных фрагментах текста программы:

```
DoCmd.OpenQuery "ТипИздДоб"
Me![ТипИздДоп] = Null
 Me![Обозн изделий].SetFocus
```

9. Из каких этапов состоит процесс выполнения операторов SQL?
10. Какие функции выполняют хранимые процедуры и на каком языке программирования в SQL Server они пишутся?
11. Какая команда применяется для написания триггера?
12. Что означает процесс оптимизации запросов?
13. Что означает параметр WITH ENCRIPTING ?
14. Назовите правила, ограничивающие состав операторов триггера.

Глава 9

ВСТРОЕННЫЙ ЯЗЫК SQL

9.1. Назначение и особенности встроенного языка SQL

В гл. 8 мы изложили возможности и технологию разработки управляющих программ в среде VBA, а в этой главе рассмотрим некоторые возможности управления базами данных средствами языка SQL.

Язык SQL (см. гл. 5) предназначен для организации доступа к базам данных. При этом предполагается, что доступ к БД может быть осуществлен в двух режимах: в интерактивном режиме и в режиме выполнения прикладных программ (приложений).

Благодаря этой двойственности SQL обладает следующими положительными качествами:

- все возможности интерактивного языка запросов доступны в прикладном программировании;
- в интерактивном режиме можно отладить основные алгоритмы обработки информации, которые в дальнейшем могут быть готовыми вставлены в работающие приложения.

SQL, хотя и является языком программирования, в силу своей специфической направленности не обладает многими возможностями универсальных языков программирования. В нем отсутствуют традиционные операторы, организующие циклы, позволяющие объявить и использовать внутренние переменные, организовать анализ некоторых условий и возможность изменения хода программы в зависимости от выполненного условия. В общем случае SQL можно назвать подъязыком, который служит исключительно для управления базами данных. Для создания приложений, настоящих программ необходимо использовать другие, базовые языки программирования, в которые операторы языка SQL будут встраиваться.

Базовыми языками программирования могут быть следующие языки: C, COBOL, PL/1, Pascal. Существуют два способа применения языков программирования в прикладных программах:

- встроенный SQL. При таком подходе операторы SQL встраиваются непосредственно в исходный текст программы на базовом языке. При компиляции программы со встроенными операторами SQL используется специальный препроцессор SQL, который преобразует исходный текст в исполняемую программу;
- языки программирования интерфейса (API — application program interface). При использовании данного способа прикла-

ная программа взаимодействует с СУБД путем применения специальных функций, вызывая эти функции.

Процесс выполнения операторов SQL может быть условно разделен на пять этапов (рис. 9.1).

На *первом* этапе выполняется синтаксический анализ оператора SQL. На этом этапе проверяется корректность записи оператора SQL в соответствии с правилами синтаксиса.

На *втором* этапе проверяется корректность параметров оператора SQL: имен отношений, имен полей данных, привилегий пользователя по работе с указанными объектами. На этом этапе отыскиваются семантические ошибки.

На *третьем* этапе проводится оптимизация запроса. СУБД проводит разделение целостного запроса на ряд минимальных операций и оптимизирует последовательность их выполнения с точки зрения временных затрат на выполнения запроса. На этом этапе строится несколько планов выполнения запроса и выбирается из них один — оптимальный для данного состояния БД.

На *четвертом* этапе СУБД генерирует двоичную версию оптимального плана опроса, подготовленного на третьем этапе. Двоичный план выполнения запроса СУБД фактически является эквивалентом объектного кода программы.

На *пятом* этапе СУБД реализует разработанный план, тем самым выполняя запрос.

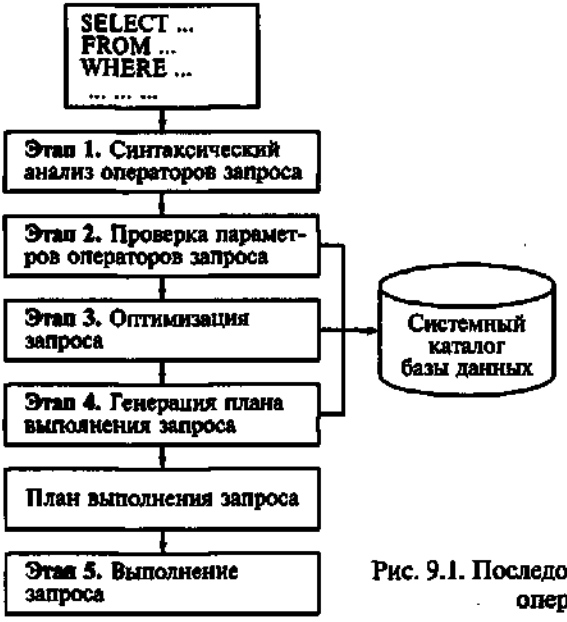


Рис. 9.1. Последовательность выполнения операторов SQL

Перечисленные этапы отличаются по числу обращений к БД и по процессорному времени, требуемому для их выполнения.

Синтаксический анализ проводится очень быстро, он не требует обращения к системным каталогам БД.

Семантический анализ уже требует работы с базой метаданных, т.е. с системными каталогами БД, поэтому при выполнении этого этапа происходит обращение к системному каталогу.

Этап, связанный с оптимизаций плана запроса, требует работы не только с системным каталогом, но и со статистической информацией о БД, которая характеризует текущее состояние всех отношений, используемых в запросе, их физическое расположение на страницах и сегментах внешней памяти. В силу указанных причин этап оптимизации наиболее трудоемкий и длительный в процессе выполнения запроса. Однако если не проводить этап оптимизации, то время выполнения неоптимизированного запроса может в несколько раз превысить время оптимизированного запроса. Время, затраченное на оптимизацию запроса, значительно компенсирует затраты на выполнение неоптимизированного запроса.

Этапы выполнения операторов SQL одни и те же, как в интерактивном режиме, так и внутри приложений.

Как правило, в различных программных системах (СУБД) разработчики встраивают собственные модификации языка SQL.

Встроенный SQL имеет свои особенности. Однако при объединении операторов SQL с базовым языком программирования должны соблюдаться следующие принципы:

- операторы SQL включаются непосредственно в текст программы на исходном языке программирования. Исходная программа поступает на вход препроцессора SQL, который компилирует операторы SQL;
- встроенные операторы SQL могут ссылаться на переменные базового языка программирования;
- встроенные операторы SQL получают результаты SQL-запросов с помощью переменных базового языка программирования;
- для присвоения неопределенных значений (NULL) атрибутам отношений БД используются специальные функции;
- для обеспечения построчной обработки результатов запросов во встроенный SQL добавляются несколько новых операторов, которые отсутствуют в интерактивном SQL.

Операторы манипулирования данными не требуют изменения для их встраивания в программный код SQL. Однако оператор поиска (SELECT) потребовал изменений.

Стандартный оператор SELECT возвращает набор данных, идентичный сформированным условиям запроса. В интерактивном SQL этот полученный набор данных просто выводится на консоль пользо-

вателя и он может просмотреть полученные результаты. Встроенный оператор SELECT должен создавать структуры данных, которые согласуются с базовыми языками программирования.

Во встроенном SQL запросы делятся на два типа:

• однострочные запросы, где ожидаемые результаты соответствуют одной строке данных. Эта строка может содержать значения нескольких столбцов;

• многострочные запросы, результатом которых является получение целого набора строк. При этом приложение должно иметь возможность проработать все полученные строки. Значит, должен существовать механизм, который поддерживает просмотр и обработку полученного набора строк.

Первый тип запроса — однострочный запрос во встроенном SQL — вызвал модификацию оператора SQL, которая выглядит следующим образом:

```
SELECT [{ALL | DISTINCT}] <список возвращаемых столбцов>
INTO <список переменных базового языка>
FROM <список исходных таблиц>
[WHERE <условия соединения и поиска>]
```

Во встроенный оператор SELECT добавился новый оператор INTO, содержащий список переменных базового языка. Именно в эти переменные будет помещен результат однострочного запроса, поэтому список переменных базового языка должен быть согласован как по порядку, так и по типу и размеру данных со списком возвращаемых столбцов. По правилам любого языка программирования все базовые переменные предварительно описаны в прикладной программе. Например, если в БД «Библиотека» существует таблица READERS («Читатели»), то мы можем получить сведения о конкретном читателе.

```
CREATE TABLE READERS
(READER_JD Smallint (4) PRIMARY KEY.
FIRST_NAME char (30) NOT NULL.
LAST_NAME char (30) NOT NULL.
ADRES char (50).
HOME_PHON char (12).
WORK_PHON char (12).
BIRTH_DAY date CHECK (DateDiff (year. GetDate () DIRTH_DAY)>=17
):
```

Для этого опишем базовые переменные. Рассмотрим пример для MS SQL SERVER 7.0, используя язык Transact SQL. При описании локальных переменных в языке Transact SQL используется специальный символ «@». Комментарии в Transact SQL заключены в парные символы /* комментарий */.

```
DECLARE @reader_id INT
DECLARE @FIRS_NAME Char (30), @LAST_NAME Char (50).
@ADRES Char (50)
DECLARE @HOME_PHON Char(12).13WORK_PHON Char(12)
/* зададим уникальный номер читательского билета */
SET @READER_ID = 4
/* теперь выполним запрос и поместим полученные сведения
в определенные ранее переменные */
SELECT READERS.FIRST_NAME. READERS.IAST_NAME.
READERS.ADRES.
READERS.HOME_PHON. READERS.WORK_PHON
INTO @FIRS_NAME. @LAST_NAME. @ADRES. @HOME_PHON.
@WORK_PHON
FROM READERS
WHERE READERS.READER_ID = @READER_ID
```

В этом простом примере мы имена переменных сделали такими же, как и имена столбцов таблицы READERS, но это необязательно. Однако транслятор различат эти объекты, именно поэтому в диалекте Transact SQL принято локальные переменные предварять специальным символом «@». В примере мы использовали квалифицированные имена полей, имена полей, предваряемые именем таблиц. Однако это необязательно, потому что запрос выбирает данные только из одной таблицы.

В данном примере базовые переменные играют разную роль. Локальная переменная @READER_ID является входной по отношению к запросу. Ей присвоено значение 4. В запросе это значение используется для фильтрации данных, поэтому эта переменная используется в условии WHERE.

Остальные базовые переменные играют роль выходных переменных. В них СУБД помещает результат выполнения запроса, помещая в них значения соответствующих полей отношения READERS, извлеченные из БД.

9.2. Курсоры — операторы обработки многострочных запросов

Рассмотрим более сложные запросы — многострочные. Для реализации многострочных запросов вводится понятие курсора, или указателя набора записей.

Для работы с курсором добавляется несколько новых операторов SQL:

DECLARE CURSOR определяет выполняемый запрос, задает имя курсора и связывает результаты запроса с заданным курсором. Этот оператор не является исполняемым для запроса, он только определяет структуру будущего множества записей и связывает

ее с уникальным именем курсора. Этот оператор подобен операторам описания данных в языках программирования;

OPEN дает команду СУБД выполнить описанный запрос, создать виртуальный набор строк, который соответствует заданному запросу. Оператор OPEN устанавливает указатель записей (курсор) перед первой строкой виртуального набора строк результата;

FETCH продвигает указатель записей на следующую позицию. Во многих СУБД этот оператор реализует более широкие функции перемещения. Он позволяет перемещать указатель па произвольную запись, вперед и назад, допускает как абсолютную адресацию, так и относительную адресацию, позволяет установить курсор на первую или последнюю запись динамической таблицы;

CLOSE закрывает курсор и прекращает доступ к записям динамической таблицы. Он фактически ликвидирует связь между курсором и результатом выполнения базового запроса. Однако в коммерческих СУБД оператор CLOSE не всегда означает уничтожение виртуального набора записей.

Стандарт определяет следующий синтаксис оператора определения курсора:

DECLARE <имя_курсора> CURSOR FOR
<спецификация_курсора> <спецификация курсора>::= <выражение_запроса SELECT>

Имя курсора — это допустимый идентификатор в базовом языке программирования.

В объявлении курсора могут быть использованы базовые переменные. Однако необходимо помнить, что на момент выполнения оператора OPEN значения всех базовых переменных, используемых в качестве входных переменных, связанных с условиями фильтрации значений в базовом запросе, должны быть уже заданы.

Определим курсор, который содержит список всех должников библиотеки. Должниками назовем читателей, которые имеют на руках хотя бы одну книгу, срок сдачи которой уже прошел.

```
DECLARE Debtor_reader_cursor CURSOR FOR
SELECT READERS. FIRST_NAME. READERS. LAST_NAME,
READERS.ADRES.
READERS.HOME_PHON. READERS.WORK_PHON. BOOKS.
TITLE
FROM READERS. BOOKS.EXEMPLAR
WHERE READERS.READER_ID = EXEMPLAR. READER_ID
AND
BOOKS.ISBN = EXEMPLARE. ISBN AND
EXEMPLAR. DATA_JUT > Getdate()
ORDER BY READERS. FIRST_NAME
```

При определении курсора мы снова использовали функцию Transact SQL Getdate(), которая возвращает значение текущей даты. Таким образом, определенный курсор будет создавать набор строк, содержащих перечень должников, с указанием названий книг, которые они не вернули вовремя в библиотеку.

В соответствии со стандартом SQL2 Transact SQL содержит расширенное определение курсора

DECLARE <имя_курсора> [INSENSITIVE] [SCROLL] CURSOR FOR<оператор выбора SELECT>
[FOR {READ ONLY | UPDATE [OF <имя_столбца 1> [....n]]}]

Параметр INSENSITIVE (нечувствительный) определяет режим создания набора строк, соответствующего определяемому курсору, при котором все изменения в исходных таблицах, произведенные после открытия курсора другими пользователями, не видны в нем. Такой набор данных нечувствителен ко всем изменениям, которые могут проводиться другими пользователями в исходных таблицах, этот тип курсора соответствует некоторому «мгновенному слепку» с БД.

СУБД более быстро и экономно может обрабатывать такой курсор. Поэтому если для вас действительно важно рассмотреть и обработать состояние БД на некоторый конкретный момент времени, то имеет смысл создать «нечувствительный курсор».

Ключевое слово SCROLL определяет, что допустимы любые режимы перемещения по курсору (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) в операторе FETCH.

Если не указано ключевое слово SCROLL, то считается доступной только стандартное перемещение вперед: спецификация NEXT в операторе FETCH.

Если указана спецификация READ ONLY (только для чтения), то изменения и обновления исходных таблиц не будут выполняться с использованием данного курсора. Курсор с данной спецификацией может быть самым быстрым в обработке. Однако если вы не укажите специально спецификацию READ ONLY, то СУБД будет считать, что вы допускаете операции модификации с базовыми таблицами, и в этом случае для обеспечения целостности БД система будет гораздо медленнее обрабатывать операции с курсором.

При использовании параметра UPDATE [OF <имя столбца 1> [,...<имя столбца n>]] мы задаем перечень столбцов, в которых допустимы изменения в процессе нашей работы с курсором. Такое ограничение упростит и ускорит работу СУБД. Если этот параметр не указан, то предполагается, что допустимы изменения всех столбцов курсора. Вернемся к нашему примеру. Если мы преследуем цель мгновенного слепка БД, дающего сведения о должниках, то применим все параметры, позволяющие ускорить рабо-

ту с курсором. Тогда оператор описания курсора будет выглядеть следующим образом:

```
DECLARE Debtor_reader_cursor INSENSITIVE CURSOR
FOR
SELECT READERS.FIRST_NAME. READERS.LAST_NAME.
READERS.ADRES. READERS.HOME_PHON. READERS.WORK_
PHON. BOOKS.TITLE
FROM READERS.BOOKS.EXEMPLAR
WHERE READERS.READER_ID = EXEMPLAR.READER_ID
AND
BOOKS.ISBN = EXEMPLARE.ISBN AND
EXEMPLAR.DATA_OUT > Getdate ()
ORDER BY READERS.FIRST_NAME
FOR READ ONLY
```

При описании курсора нет ограничений на вид оператора SELECT, который используется для создания базового набора строк, связанного с курсором. В операторе SELECT могут использоваться группировки и встроенные подзапросы, и вычисляемые поля.

Расширенный оператор FETCH имеет следующий синтаксис:

```
FETCH
[NEXT | PRIOR | FIRST | LAST
|ABSOLUTE {n | <имя_переменной>}
| RELATIVE{n|< имя_переменной >}]
FROM
<имя курсора> INTO <список базовых переменных>
```

Здесь параметр NEXT задает выбор следующей строки после текущей из базового набора строк, связанного с курсором. Параметр PRIOR задает перемещение на предыдущую строку по отношению к текущей. Параметр FIRST задает перемещение на первую строку набора, а параметр LAST задает перемещение на последнюю строку набора.

Кроме того, в расширенном операторе перемещения допустимо переместиться сразу на заданную строку; при этом допустима как абсолютная адресация — заданием параметра ABSOLUTE, так и относительная адресация — заданием параметра RELATIVE. При относительной адресации положительное число сдвигает указатель вниз от текущей записи, отрицательное число сдвигает указатель вверх от текущей записи.

Однако для применения расширенного оператора FETCH в соответствии со стандартом SQL2 описание курсора обязательно должно содержать ключевое слово SCROLL. Иногда такие курсоры называют прокручиваемыми. В стандарт эти курсоры вошли сравнительно недавно, поэтому в коммерческих СУБД очень час-

то операторы по работе с подобными курсорами сильно отличаются. Правда, реалии сегодняшнего дня заставляют поставщиков коммерческих СУБД более строго соблюдать последний стандарт SQL. В технической документации можно встретить две версии синтаксиса оператора FETCH: одну, которая соответствует стандарту, и другую, которая расширяет стандарт дополнительными возможностями, предоставляемыми только данной СУБД для работы с курсором.

Если вы предполагаете, что БД должна быть перенесена на другую платформу, а это надо всегда предусматривать, то лучше пользоваться стандартными возможностями. В этом случае приложение будет более платформенно-независимым и его легче будет перенести на другую СУБД.

9.3. Оператор закрытия курсора

Оператор закрытия курсора имеет простой синтаксис, он выглядит следующим образом:

CLOSE <имя_курсора>

Оператор закрытия курсора закрывает временную таблицу, созданную оператором открытия курсора, и прекращает доступ прикладной программы к этому объекту. Единственным параметром оператора закрытия является имя курсора.

Оператор закрытия курсора может быть выполнен в любой момент после оператора открытия курсора.

В некоторых коммерческих СУБД кроме оператора закрытия курсора используется еще оператор деактивации (уничтожения) курсора. Например, в MS SQL Server 7.0 наряду с оператором закрытия курсора используется оператор

DEALLOCATE <имя_курсора>

Здесь оператор закрытия курсора не уничтожает набор данных, связанный с курсором, а только закрывает к нему доступ и освобождает все блокировки, которые ранее были связаны с данным курсором.

При выполнении оператора DEALLOCATE SQL Server освобождает разделяемую память, используемую командой описания курсора DECLARE. После выполнения этой команды невозможно выполнение команды OPEN для данного курсора.

9.4. Удаление и обновление данных с использованием курсора

Курсоры в прикладных программах часто используются для последовательного просмотра данных. Если курсор не связан с операцией группировки, то фактически каждая строка курсора соответствует строго только одной строке исходной таблицы и в

этом случае курсор удобно использовать для оперативной корректировки данных. В стандарте определены операции модификации данных, связанные с курсором. Операция удаления строки, связанной с текущим указателем курсора, имеет следующий синтаксис:

DELETE FROM <имя__таблицы> WHERE CURRENT OF <имя курсора>

Если указанный в операторе курсор открыт и установлен на некоторую строку и курсор определяет изменяемую таблицу, то текущая строка курсора удаляется, а он позиционируется перед следующей строкой. Таблица, указанная в разделе FROM оператора DELETE, должна быть таблицей, указанной в самом внешнем разделе FROM спецификации курсора.

Если необходимо прочитать следующую строку курсора, то надо снова выполнить оператор FETCH NEXT.

Аналогично курсор может быть использован для модификации данных. Синтаксис операции позиционной модификации следующий:

UPDATE <имя_таблицы> SET <имя_столбца1>= (<значение> | NULL} [{,<имя_столбца_N>= {<значение> | NULL}}...]
WERE CURRENT OF <имя_курсора>

Одним оператором позиционного обновления могут быть заменены несколько значений столбцов строки таблицы, соответствующей текущей позиции курсора. После выполнения операции модификации позиция курсора не изменяется.

Для того чтобы можно было применять позиционные операторы удаления (DELETE) и модификации (UPDATE), курсор, согласно стандарту SQL1, должен удовлетворять следующим требованиям:

- запрос, связанный с курсором, должен считывать данные из одной исходной таблицы, т.е. в предложении FROM запроса SELECT, связанного с определением курсора (DECLARE CURSOR), должна быть задана только одна таблица;
- в запросе не может присутствовать параметр упорядочения ORDER BY. Для того чтобы сохранялось взаимно-однозначное соответствие строк курсора и исходной таблицы, курсор не должен идентифицировать упорядоченный набор данных;
- в запросе не должно присутствовать ключевое слово DISTINCT;
- запрос не должен содержать операций группировки, т.е. в нем не должно присутствовать предложение GROUP BY или HAVING;
- пользователь, который хочет применить операции позиционного удаления или обновления, должен иметь соответствующие права на выполнение данных операций над базовой таблицей.

Использование курсора для операций обновления значительно усложняет работу с подобным курсором со стороны СУБД, поэтому операции, связанные с позиционной модификацией, выполняются гораздо медленнее, чем операции с курсорами, которые используются только для чтения. Именно поэтому рекомендуется обязательно указывать в операторе определения курсора предложение READ ONLY, если вы не собираетесь использовать данный курсор для операций модификации. По умолчанию, если нет дополнительных указаний, СУБД создает курсор с возможностью модификации.

Курсоры — удобное средство для формирования бизнес-логики приложений, но следует помнить, что если вы открываете курсор с возможностью модификации, то СУБД блокирует все строки базовой таблицы, вошедшие в ваш курсор, и тем самым блокируется работа других пользователей с данной таблицей.

Чтобы свести к минимуму число требуемых блокировок, при работе интерактивных программ следует придерживаться следующих правил:

- необходимо делать транзакции как можно короче;
- необходимо выполнять оператор завершения COMMIT после каждого запроса и как можно скорее после изменений, сделанных программой;
- необходимо избегать программ, в которых осуществляется интенсивное взаимодействие с пользователем или осуществляется просмотр очень большого числа строк данных;
- если возможно, то лучше не применять прокручиваемые курсоры (SCROLL), потому что они требуют блокирования всех строк выборки, связанных с открытым курсором;
- использование простого последовательного курсора позволит системе разблокировать текущую строку, как только будет выполнена операция FETCH, что минимизирует блокировки других пользователей, работающих параллельно с вами и использующих те же таблицы;
- если возможно, определяйте курсор как READ ONLY.

Когда мы рассматривали модели клиент—сервер, то определили, что в развитых моделях серверов баз данных большая часть бизнес-логики клиентского приложения выполняется именно на сервере, а не на клиенте. Для этого используются специальные объекты, которые называются хранимыми процедурами и хранятся в БД, как таблицы и другие базовые объекты.

В связи с этим фактом курсоры, которые могут быть использованы в приложениях, обычно подразделяются на курсоры сервера и курсоры клиента.

Курсор сервера создается и выполняется на сервере; данные, связанные с ним, не пересылаются на компьютер клиента. Курсоры сервера определяются обычно в хранимых процедурах или триггерах.

Курсоры клиента — это те курсоры, которые определяются в прикладных программах, выполняемых на клиенте. Набор строк, связанный с данным курсором, пересылается на компьютер клиента и там обрабатывается. Если с курсором связан большой набор данных, то операция пересылки набора строк, связанных с курсором, может занять значительное время и значительные ресурсы сети и клиентского компьютера.

Конечно, курсоры сервера более экономичны и выполняются быстрее. Поэтому последней рекомендацией, связанной с использованием курсоров, будет рекомендация трансформировать логику работы вашего приложения, чтобы как можно чаще вместо курсоров клиента использовать курсоры сервера.

9.5. Хранимые процедуры

С точки зрения приложений, работающих с БД, хранимые процедуры (Stored Procedure) — это подпрограммы, которые выполняются на сервере. По отношению к БД — это объекты, которые создаются и хранятся в БД. Они могут быть вызваны из клиентских приложений. При этом одна процедура может быть использована в любом количестве клиентских приложений, что позволяет существенно сэкономить трудозатраты на создание прикладного программного обеспечения и эффективно применять стратегию повторного использования кода. Так же как и любые процедуры в стандартных языках программирования, хранимые процедуры могут иметь входные и выходные параметры или не иметь их вообще.

Хранимые процедуры могут быть активизированы не только пользовательскими приложениями, но и триггерами.

Хранимые процедуры пишутся на специальном встроенном языке программирования. Они могут включать в себя любые операторы SQL, а также включают в себя некоторый набор операторов, управляющих ходом выполнения программ, которые во многом схожи с подобными операторами процедурно-ориентированных языков программирования. В коммерческих СУБД для написания текстов хранимых процедур используются собственные языки программирования. Так, в СУБД Oracle для этого используется язык PL/SQL, а в MS SQL Server используется язык Transact SQL. В последних версиях Oracle объявлено использование языка Java для написания хранимых процедур.

Хранимые процедуры являются объектами БД. Каждая хранимая процедура компилируется при первом выполнении. В процессе компиляции строится оптимальный план выполнения процедуры. Описание процедуры совместно с планом ее выполнения хранится в системных таблицах БД.

Для создания хранимой процедуры применяется оператор SQL CREATE PROCEDURE.

По умолчанию выполнить хранимую процедуру может только ее владелец, которым является владелец БД, и создатель хранимой процедуры. Однако владелец хранимой процедуры может делегировать права на ее запуск другим пользователям.

Имя хранимой процедуры является идентификатором в языке программирования, на котором она пишется, и должно удовлетворять всем требованиям, которые предъявляются к идентификаторам в данном языке.

В MS SQL Server хранимая процедура создается следующим оператором:

```
CREATE PROCEDURE <имя_процедуры> [;<версия>]
{(@параметр1тип_данных)
[VARYING] [=<значение_по_умолчанию>] [OUTPUT]|
[,.параметрN.]
[ WITH
{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}]
[FOR REPLICATION]
AS Тело процедуры
```

Здесь необязательное ключевое слово VARYING определяет заданное значение по умолчанию для определенного ранее параметра.

Ключевое слово RECOMPILE определяет режим компиляции создаваемой хранимой процедуры. Если задано ключевое слово RECOMPILE, то процедура будет перекомпилироваться каждый раз, когда она будет вызываться на исполнение. Это может резко замедлить исполнение процедуры. Но если данные, обрабатываемые данной хранимой процедурой, настолько динамичны, что предыдущий план исполнения, составленный при ее первом вызове, может быть абсолютно неэффективен при последующих вызовах, то стоит применять данный параметр при создании этой процедуры.

Ключевое слово ENCRYPTION определяет режим, при котором исходный текст хранимой процедуры не сохраняется в БД. Такой режим применяется для того, чтобы сохранить авторское право на интеллектуальную продукцию, которой и являются хранимые процедуры. Часто такой режим применяется тогда, когда разработчик не хочет, чтобы исходные тексты разработанных процедур были доступны администратору БД, работающему у заказчика.

Однако кроме имени хранимой процедуры все остальные параметры являются необязательными. Процедуры могут быть процедурами или процедурами-функциями. Эти понятия здесь трактуются традиционно, как в языках программирования высокого уровня. Хранимая процедура-функция возвращает значение, которое присваивается переменной, определяющей имя процедуры.

Процедура в явном виде не возвращает значение, но в ней может быть использовано ключевое слово OUTPUT, которое определяет, что данный параметр является выходным.

Рассмотрим несколько примеров простейших хранимых процедур.

```
/* процедура проверки наличия экземпляров конкретной книги в библиотеке
параметры:
@ISBN — шифр книги — процедура возвращает параметр, равный количеству экземпляров. Если возвращается ноль, то это значит, что нет свободных экземпляров данной книги в библиотеке. */
CREATE PROCEDURE COUNT_EX ((@ISBN varchar (12))
AS
/* определим внутреннюю переменную */
DECLARE (@TEK_COUNT int
/* выполним соответствующий оператор SELECT

Будем считать только экземпляры, которые в настоящий момент находятся не на руках у читателей, а в библиотеке */

SELECT @TEK_COUNT = select count (*)
FROM EXEMPLAR
WHERE ISBN = @ISBN
AND READER_@ID Is NULL AND EXIST = True
/* 0 — ноль означает, что нет ни одного свободного экземпляра данной книги в библиотеке */
RETURN @TEK_COUNT
```

Хранимая процедура может быть вызвана несколькими способами. Простейший способ — это использование оператора:

```
EXEC <имя процедуры> <значение_входного_параметра1>... <имя_переменной_для_выходного параметра 1>...
```

При этом все входные и выходные параметры должны быть заданы обязательно в том порядке, в котором они определены в процедуре.

Например, если нужно найти число экземпляров книги «Oracle8. Энциклопедия пользователя», которая имеет ISBN 966-7393-08-09, то текст вызова ранее созданной хранимой процедуры может быть следующим:

```
/*Определили две переменные: @Ntek — количество экземпляров данной книги в наличии в библиотеке; @ISBN — международный шифр книги */
declare @Ntek int
DECLARE @ISBN VARCHAR(14)
/* Присвоим значение переменной @ISBN */
```

```
SELECT @ISBN = '966-7393-08-09'
/* Присвоим переменной @Ntek результаты выполнения хранимой процедуры COUNT_EX */
EXEC @Ntek =COUNT_EX:2 @ISBN
```

Если определено несколько версий хранимой процедуры, то при вызове вы можете указать номер конкретной версии для исполнения. Например, в версии 2 процедуры COUNT_EX последний оператор исполнения этой процедуры имеет вид

```
EXEC @Ntek =COUNT_EX:2 @ISBN
```

Однако если в процедуре определены значения входных параметров по умолчанию, то при запуске процедуры могут быть указаны значения не всех параметров. В этом случае оператор вызова процедуры может быть записан в следующем виде:

EXEC <имя процедуры> <имя_параметра1>=<значение параметра 1>... <имя_параметраN>=<значение параметраN>...

Например, создадим процедуру, которая считает количество книг, изданных конкретным издательством в конкретном году. При создании процедуры зададим для года издания по умолчанию значение текущего года.

```
CREATE PROCEDURE COUNT_BOOKS (@YEARIZD int =
Year (GetDate()).
@PUBLICH varchar (20))
/* процедура подсчета количества книг конкретного издательства, изданных в конкретном году. Параметры: @YEARIZD int год издания и
@PUBLICH название издательства */
AS
DECLARE @TEK_Count int
SELECT @TEK_COUNT = SELECT COUNT (ISBN)
FROM BOOKS
WHERE YEARIZD =@YEARIZD AND PUBLICH = @PUBLICH
Одновременно с исполнением оператора SELECT результаты его выполнения присваиваются определенной ранее переменной @ILK_Count */
/* при формировании результата работы процедуры мы должны учесть, что в библиотеке, возможно, нет ни одной книги некоторого издательства для заданного года. Результат выполнения запроса SELECT в этом случае будет иметь неопределенное значение, но анализировать все-таки лучше числовые значения. Поэтому в качестве возвращаемого значения мы используем результаты работы специальной встроенной функции языка Transact SQL COALESCE (nl,n2,...,nm), которая возвращает первое конкретное значение из списка значений nl,n2,...,nm, т.е. не равное NULL */
```

```
RETURN COALESCE (@TEK_Count, ())
```

Теперь вызовем эту процедуру, для чего подготовим переменную, в которую можно поместить результаты выполнения процедуры:

```
declare @N int
```

В переменной @N мы получим число книг в библиотеке, изданных издательством «АКАДЕМИЯ» в текущем году. Мы можем обратиться к этой процедуре, задав все параметры:

```
EXEC @N = COUNT_BOOKS @PUBLICH = 'АКАДЕМИЯ',
@YEARIZD = 2004
```

Тогда получим число книг, изданных издательством «АКАДЕМИЯ» в 2004 г. и присутствующих в нашей библиотеке.

Если мы задаем параметры по именам, то нам необязательно задавать их в том порядке, в котором они описаны при создании процедуры.

Каждая хранимая процедура является объектом БД. Она имеет уникальное имя и уникальный внутренний номер в системном каталоге. При изменении текста хранимой процедуры мы должны сначала удалить данную процедуру как объект, хранимый в БД, и только после этого записать на ее место новую. Следует отметить, что при удалении хранимой процедуры удаляются одновременно все ее версии.

Для того чтобы автоматизировать процесс удаления старой процедуры и замены ее на новую, в начале текста хранимой процедуры можно выполнить проверку наличия объекта типа «хранимая процедура» с данным именем в системном каталоге и при наличии описания данного объекта удалить его из системного каталога. В этом случае текст хранимой процедуры предваряется специальным оператором проверки и может иметь, например, следующий вид:

```
/* проверка существования в системном каталоге объекта с
данным именем и типом, созданного владельцем БД */
IF exists (select * from sysobjects where id = object_id
('dbo.NEW_BOOKS') and systant & Oxf = 4)
/* если объект существует, то сначала его удалим из системного каталога*/
drop procedure dbo.NEW_BOOKS
GO
CREATE PROCEDURE NEW_BOOKS (@ISBN varchar
(12).@TITL varchar (255),@AUTOR varchar (30).@COAUTOR
varchar (30).@
YEARIZD int. @PAGES INT,@NUM_EXEMPL INT)
/* процедура ввода новой книги с указанием количества экземпляров данной книги
```

```
Параметры
@ISBN          varchar (12) — шифр книги
@TITL          varchar(255) — название
@AUTOR         varchar(30)  — автор
@COAUTOR       varchar(30)  — соавтор
@YEARIZD       int          — год издания
@PAGES         int          — количество страниц
@NUM_EXEMPL    int          — количество экземпляров */
AS
/*опишем переменную, в которой будет храниться количество оставшихся не оприходованных экземпляров книги, т.е. таких, которым еще не заданы инвентарные номера */
/DECLARE @TEK int
/ *вводим данные о книге в таблицу BOOKS */
INSERT INTO BOOKS VALUES @ISBN. @TITL. @AUTOR. @COAUTOR.@YEARIZD@PAGES)
/* назначение значения текущего счетчика оставшихся к вводу экземпляров /
SELECT @TEK = @NUM_EXEMPL
/* организуем цикл для ввода новых экземпляров данной книги */ WHILE @TEK>0 /* пока количество оставшихся экземпляров больше нуля */
BEGIN
/* так как для инвентарного номера экземпляра книги мы задали свойство IDENTITY, то нам не надо вводить инвентарный номер. СУБД сама автоматически вычислит его, добавив единицу к предыдущему, введет при выполнении оператора ввода INSERT.
Поле, определяющее присутствие экземпляра в библиотеке (EXIST), — логическое поле; мы введем туда значение TRUE, которое соответствует присутствию экземпляра книги в библиотеке */
INSERT into EXEMPLAR (ISBN.DATA_IN. DATA_OUT. EXIST)
VALUES (@ISBN.GetDate ().GetDate ()).TRUE)
/* изменение текущего значения счетчика количества оставшихся экземпляров */
SELECT @TEK = @TEK – 1
END /* конец цикла ввода данных о экземпляре книги*/
GO
```

Если бы мы не использовали инкрементное поле в качестве инвентарного номера экземпляра, то мы могли бы сами назначать инвентарный номер, увеличивая на единицу номер последнего хранимого в библиотеке экземпляра книги. Можно было бы попробовать просто сосчитать количество существующих экземпляров в библиотеке, но мы могли бы удалить некоторые. Тогда номер нового экземпляра может быть уже использован и мы не

сможем ввести данные, система не позволит нам нарушить уникальность первичного ключа.

Текст процедуры в этом случае будет иметь следующий вид:

```
/* проверка существования в системном каталоге объекта с данным именем и типом, созданного владельцем БД */
if exists (select * from sysobjects where id = object_id('dbo. NEW_BOOKS') and sysstat & Oxf = 4)
/* если обьект существует, то сначала его удалим из системного каталога*/
drop procedure dbo. NEW_BOOKS
CREATE PROCEDURE NEW_BOOKS (@ISBN varchar (12).@TITL varchar (255),@AUTOR varchar (30).@COAUTOR varchar (30).@
YEARIZD int. @PAGES INT,@NUM_EXEMPL INT)
/* процедура ввода новой книги с указанием количества экземпляров данной книги
Параметры
@ISBN          varchar (12) — шифр книги
@TITL          varchar(255) — название
@AUTOR         varchar(30)  — автор
@COAUTOR       varchar(30)  — соавтор
@YEARIZD       int          — год издания
@PAGES         int          — количество страниц
@NUM_EXEMPL    int          — количество экземпляров */
*/
DECLARE @TEK int
DECLARE @INV int
INSERT INTO BOOKS VALUES (@ISBN.@TITL.@AUTOR. @COAUTOR.@YEARIZD.@PAGES)
/* задание значения текущего счетчика оставшихся к вводу экземпляров */
SELECT @TEK = @NUM_EXEMPL
/* определение максимального значения инвентарного номера в библиотеке */
SELECT @INV = SELECT MAX (ID_XEMPLAR)
FROM EXEMPLAR
/* организуем цикл для ввода новых экземпляров данной книги */
WHILE @TEK>0
/* пока количество оставшихся экземпляров больше нуля */
BEGIN
insert into EXEMPLAR (ID_XEMPLAR.ISBN.DATA_IN. DATA_OUT.EXIST)
VALUES ((@INV.(@ISBN.GETDATE (). GETDATE (), TRUE)
/* изменение текущих значений счетчика и инвентарного номера */
```

```
SELECT @TEK = @TEK – 1
SELECT @INV = @INV + 1
END
/* конец цикла ввода данных о экземпляре книги */
GO
```

Хранимые процедуры могут вызывать одна другую. Создадим хранимую процедуру, которая возвращает номер читательского билета для конкретного читателя:

```
if exists (select * from sysobjects where id = object fd ('dbo.
CK_READER') AND sysstat & Oxf = 4)
/* если объект существует, то сначала его удалим из системного каталога */
drop procedure dbo.CK_READER
/* Процедура возвращает номер читательского билета, если читатель есть, и 0 — в противном случае. В качестве параметров передаем фамилию и дату рождения */
CREATE PROCEDURE CK_READER (@FIRST_NAME varchar (30). @BIRTH_DAY varchar (12))
AS
/*опишем переменную, в которой будет храниться номер читательского билета*/
DECLARE @NUM_READER INT
/* определение наличия читателя */
SELECT @NUM_READER = STLECT NUM_READER
FROM READERS
WHERE FIRST_NAME = @ FIRSTJAME AND convert (varchar (8). BIRTH_DAY,4) = @BIRTH_DAY
RETURN COALESCE ((@NUM_READER.0)
```

Мы здесь использовали функцию преобразования типа данных dataTime в тип данных varchar(8). Это было необходимо сделать для согласования типов данных при выполнении операции сравнения. Действительно, входная переменная @BIRTH_DAY имеет символьный тип (varchar), а поле базы данных BIRTH_DAY имеет тип SmallDateTime.

Хранимые процедуры допускают наличие нескольких выходных параметров. Для этого каждый выходной параметр должен после задания своего типа данных иметь дополнительное ключевое слово OUTPUT. Рассмотрим пример хранимой процедуры с несколькими выходными параметрами.

Создадим процедуру ввода нового читателя; при этом внутри процедуры выполним проверку наличия в нашей картотеке данного читателя, чтобы не назначать ему новый номер читательского билета. Выходными параметрами процедуры будут номер читательского билета — признак того, был ли ранее записан читатель

с данными характеристиками в нашей библиотеке, а если он был записан, то сколько книг за ним числится.

```
/* проверка наличия данной процедуры в БД*/
if exists (select * from sysobjects where id =
object_id (N'[dbo].[NEW_READER]') and OBJECTPROPERTY
(id, N' Is Procedure')=1)
drop procedure [dbo].[NEW_READER]
GO
/* процедура проверки существования читателя с заданными
значениями вводимых параметров
Процедура возвращает новый номер читательского билета, если
такого читателя не было, в противном случае — сообщает старый
номер и количество книг, которое должен читатель */
CREATE PROCEDURE NEW_READER
(@NAME_READER varchar (30), (@ADRES varchar (40).
@HOOM_PHONE char (9). (@WORK_PHONE char (9),
@BIRTH_DAY varchar (8).
@NUM_READER int OUTPUT.
/* выходной параметр, определяющий номер читательского би-
лета*/
@Y_N int OUTPUT.
/* выходной параметр, определяющий, был ли читатель ранее
записан в библиотеку */
@COUNT_BOOKS int OUTPUT
/* выходной параметр, определяющий количество книг, кото-
рое числится за читателем */)
AS
/* переменная, в которой будет храниться номер читательско-
го билета, если читатель уже был записан в библиотеку */
DECLARE @N_R int
'* определение наличия читателя */
EXEC @N_R = CK_READER @NAME_READER.@BIRTH_DAY
IF @N_R= 0 Or @N_R Is Null
* если читатель с заданными характеристиками не найден, т.е.
переменной @N_R присвоено значение нуль или ее значение не
определено, перейдем к назначению для нового читателя нового
номера читательского билета */
BEGIN

/* так как мы номер читательского билета определили как ин-
крементное поле, то в операторе ввода мы его не указываем, си-
стема сама назначит новому читателю очередной номер */
INSERT INTO RADER NAME_READER.ADRES.
HOOM_PHONE,WORK_PHONE. BIRTH_DAY)
VALUES (@NAME_READER.@ADRES.@HOOM_PHONE.
(@WORK_PHONE, Convert (smalldatetime. @BIRTH_DAY.4))
```

```
/* в операторе INSERT мы должны преобразовать символьную
переменную @BIRTH_DAY в тип данных smalldatetime, который
определен для поля BIRTH_DAY (дата рождения). Это преобразо-
вание мы сделаем с помощью встроенной функции языка Transact
SQL Convert */
/* теперь определим назначенный номер читательского биле-
та */
SELECT @NUM_READER = NUM_READER
FROM READER
WHERE NAME_READER = @NAME_READER
Convert (varchar (8).BIRTH_DAY.4) = @BIRTH_DAY
/* здесь мы снова используем функцию преобразования типа,
но в этом случае нам необходимо преобразовать поле BIRTH_DAY
из типа smalldatetime к типу varchar (8). в котором задан входной
параметр @BIRTH_DAY */
SELECT @Y_N =0
/* присваиваем выходному параметру @Y_N значение 0 (нуль),
что соответствует тому, что данный читатель ранее в нашей биб-
лиотеке не был записан */
SELECT @COUNT_BOOKS = 0
/* присваиваем выходному параметру, хранящему количество
книг, числящихся за читателем, значение нуль */
RETURN 1
END
ELSE
/* если значение переменной @N_R не равно нулю, то чита-
тель с заданными характеристиками был ранее записан в нашей
библиотеке */
BEGIN
/* определение количества книг у читателя с найденным но-
мером читательского билета */
SELECT @COUNT_BOOKS = COUNT(INV_NUMBER)
FROM EXEMPLAR WHERE NUM_READER = @N_R
SELECT @COUNT_BOOKS = COALESCE (@COUNT_
BOOKS.0)
/* присваиваем выходному параметру @COUNT_BOOKS зна-
чение, равное количеству книг, которые числятся за нашим чита-
телем; если в предыдущем запросе @COUNT_BOOKS было присво-
ено неопределенное значение, то мы заменим его на нуль, исполь-
зуя для этого встроенную функцию COALESCE(@COUNT_
BOOKS.0), которая возвращает первое определенное значение из
списка значений, заданных в качестве ее параметров */
SELECT @Y_N = 1
/* присваиваем выходному параметру @Y_N значение 1, что
соответствует тому, что данный читатель ранее в нашей библи-
отеке был записан */
```

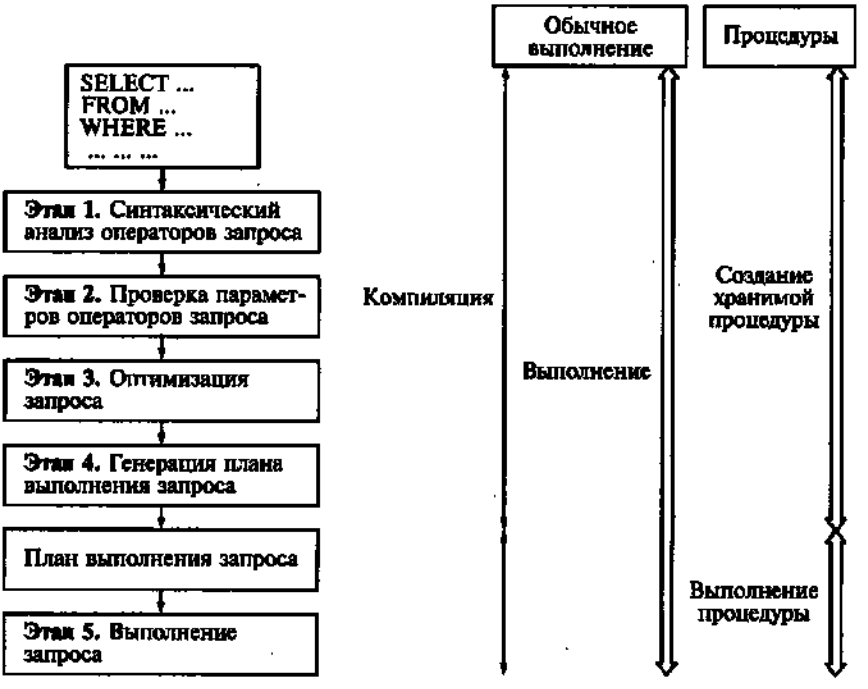


Рис. 9.2. Процесс выполнения операторов SQL на клиенте и процесс выполнения хранимой процедуры

Хранимые процедуры также играют ключевую роль в повышении быстродействия при работе в сети с архитектурой клиент—сервер.

На рис. 9.2 представлен процесс выполнения операторов SQL на «клиенте» и процесс выполнения хранимой процедуры.

В этом случае клиент обращается к серверу только для выполнения команды запуска хранимой процедуры. Сама хранимая процедура выполняется на сервере. Объем пересылаемой по сети информации резко сокращается.

9.6. Триггеры

Триггер — это специальный вид хранимой процедуры, которую SQL Server вызывает при выполнении операций модификации соответствующих таблиц. Триггер автоматически активизируется при выполнении операции, с которой он связан. Триггеры связываются с одной или несколькими операциями модификации над одной таблицей.

В разных коммерческих СУБД рассматриваются разные триггеры. Так, в MS SQL Server триггеры определены только как пост-

фильтры, т.е. такие триггеры, которые выполняются после свершения события.

В СУБД Oracle определены два типа триггеров:

- триггеры, которые могут быть запущены перед реализацией операции модификации (они называются BEFORE-триггерами);
- триггеры, которые активизируются после выполнения соответствующей модификации, аналогично триггерам MS SQL Server (они называются AFTER-триггерами).

Триггеры могут быть эффективно использованы для поддержки семантической целостности БД, однако приоритет их ниже, чем приоритет правил — ограничений (constraints), задаваемых на уровне описания таблиц и на уровне связей между таблицами. При написании триггеров всегда надо помнить об этом. При нарушении правил целостности по связям (DRI — Declarative Referential Integrity) триггер просто может никогда не сработать.

Для создания триггеров используется специальная команда:

```
CREATE TRIGGER <имя_триггера>
ON <имя_таблицы>
FOR {[INSERT][, UPDATE] [, DELETE]}
[WITH ENCRIPTING]
AS
SQL-операторы (текст программы)
```

Имя триггера является идентификатором во встроенном языке программирования СУБД и должно удовлетворять соответствующим требованиям.

В параметре FOR задается одна или несколько операций модификации, которые и запускают триггер.

Параметр WITH ENCRIPTING имеет тот же смысл, что и для хранимых процедур. Он скрывает исходный текст триггера.

Существует следующие правила, ограничивающие состав операторов триггера:

- нельзя использовать в теле триггера операции создания объектов новой БД;
- нельзя использовать в триггере команду удаления объектов DROP для всех типов объектов БД;
- нельзя использовать в теле триггера команды изменения объектов БД ALTER TABLE, ALTER DATABASE;
- нельзя изменять установленные права доступа к объектам БД, т.е. выполнять команды GRAND или REVOKE;
- нельзя создать триггер для представления (VIEW);
- в отличие от хранимых процедур триггер не может возвращать никаких значений, он запускается автоматически сервером и не может связаться самостоятельно ни с одним клиентом.

Контрольные вопросы

1. В каких двух режимах может быть осуществлен доступ к БД средствами языка SQL?
2. Какие операторы, присущие универсальным языкам программирования, отсутствуют в языке SQL?
3. На какие этапы может быть условно разделен процесс выполнения запросов операторами SQL?
4. На какие два типа подразделяются запросы во встроенном SQL?
5. Каково назначение оператора INTO?
6. Что означает понятие «курсор»?
7. Для чего используются курсоры в прикладных программах?
8. Что означают следующие операторы:
   - DECLARE CURSOR;
   - OPEN;
   - FETCH;
   - CLOSE?
9. Что означает понятие «хранимая процедура» (Stored Procedure)?
10. Какие языки программирования используются в коммерческих СУБД для написания текстов хранимых процедур?
11. Чем отличается триггер от хранимой процедуры?

ЧАСТЬ III

СИСТЕМЫ УПРАВЛЕНИЯ РАСПРЕДЕЛЕННЫМИ БАЗАМИ ДАННЫХ

Глава 10

РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

10.1. Основные понятия

При размещении СУБД на персональном компьютере, который не находится в сети, БД всегда используется в монопольном режиме. Даже если с ней работают несколько пользователей, они могут работать только последовательно.

Однако, как показала практика применения локальных баз данных, в большинстве случаев информация, которая в них содержится, носит многопользовательский характер, поэтому возникает необходимость разработки таких СУБД, которые обеспечили бы возможность одновременной работы пользователей с базами данных. Тем более, что все современные предприятия строят свою политику в области информационного обеспечения на основе принципов CALS-технологий (см. гл. 4).

Системы управления базами данных, обеспечивающие возможность одновременного доступа к информации различным пользователям называют *системами управления распределенными базами данных*. В общем случае режимы использования БД имеют вид, представленный на рис. 10.1.

Рассмотрим основные понятия, применяемые в системах управления распределенными базами данных.

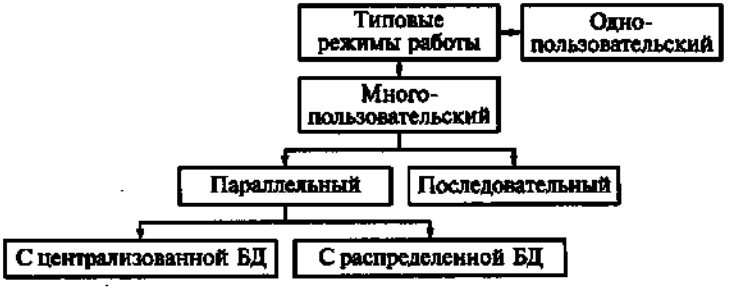


Рис. 10.1. Режимы работы с базами данных

Пользователь БД — программа или человек, обращающийся к базе данных.

Запрос — процесс обращения пользователя к БД с целью ввода, получения или изменения информации в БД.

Транзакция — последовательность операций модификации данных в БД, переводящая БД из одного непротиворечивого состояния в другое непротиворечивое состояние.

Логическая структура БД — определение БД на физически независимом уровне; ближе всего соответствует концептуальной модели БД.

Топология БД, или *структура распределенной БД*, — схема распределения физической организации базы данных в сети.

Локальная автономность означает, что информация локальной БД и связанные с ней определения данных принадлежат локальному владельцу и им управляются.

Удаленный запрос — запрос, который выполняется с использованием модемной связи.

Возможность реализации удаленной транзакции — обработка одной транзакции, состоящей из множества SQL-запросов, на одном удаленном узле.

Поддержка распределенной транзакции допускает обработку транзакции, состоящей из нескольких запросов SQL, которые выполняются на нескольких узлах сети (удаленных или локальных), но каждый запрос в этом случае обрабатывается только на одном узле.

Распределенный запрос — запрос, при обработке которого используются данные из БД, расположенные в разных узлах сети.

Системы распределенной обработки данных в основном связаны с первым поколением БД, которые строились на мультипрограммных операционных системах и использовали централизованное хранение БД на устройствах внешней памяти центральной ЭВМ и терминальный многопользовательский режим доступа. При этом пользовательские терминалы не имели собственных ресурсов, т.е. процессоров и памяти, которые могли бы использоваться для хранения и обработки данных. Первой полностью реляционной системой, работающей в многопользовательском режиме, была СУБД SYSTEM R фирмы IBM. Именно в ней были реализованы как язык манипулирования данными SQL, так и основные принципы синхронизации, применяемые при распределенной обработке данных, которые до сих пор являются базисными практически во всех коммерческих СУБД.

10.2. Модели клиент—сервер в технологии распределенных баз данных

Вычислительная модель клиент—сервер связана с появлением в 1990-х гг. открытых систем. Термин «клиент—сервер» применял-

ся к архитектуре программного обеспечения, которое состояло из двух процессов обработки информации: клиентской и серверной. Клиентский процесс запрашивал некоторые услуги, а серверный процесс обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов. Учитывая что аппаратная реализация этой модели управления базами данных связана с созданием локальных вычислительных сетей предприятия, такую организацию процесса обработки информации называют архитектурой клиент—сервер.

Основной принцип технологии клиент—сервер применительно к технологии управления базами данных заключается в разделении функций стандартного интерактивного приложения на пять групп, имеющих различную природу:

- функции ввода и отображения данных (Presentation Logic);
- прикладные функции, определяющие основные алгоритмы решения задач приложения (Business Logic);
- функции обработки данных внутри приложения (Database Logic);
- функции управления информационными ресурсами (Database Manager System);
- служебные функции, играющие роль связок между функциями первых четырех групп.

Структура типового приложения, работающего с базой данных в архитектуре клиент—сервер, приведена рис. 10.2.

Презентационная логика (Presentation Logic) как часть приложения определяется тем, что пользователь видит на своем экране, когда работает приложение. Сюда относятся все интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения. К этой же части относится все то, что выводится пользователю на экран как результаты решения некоторых промежуточных задач либо как справочная информация. Поэтому основными задачами презентационной логики являются:

- формирование экранных изображений;
- чтение и запись в информации экранные формы;
- управление экраном;

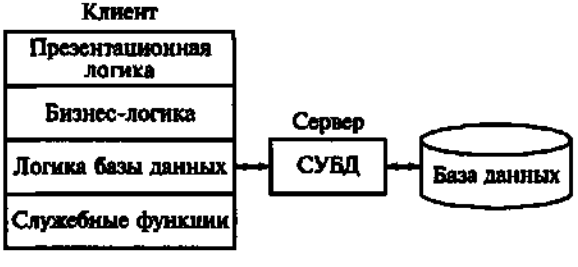


Рис. 10.2. Структура типового приложения, работающего с базой данных

• обработка движений мыши и нажатие клавиш клавиатуры.

Бизнес-логика, или *логика собственно приложений* (Business processing Logic), — это часть кода приложения, которая определяет собственно алгоритмы решения конкретных задач приложения. Обычно этот код пишется с использованием различных языков программирования, таких как C, C++, Visual-Basic и др.

Логика обработки данных (Data manipulation Logic) — это часть кода приложения, которая непосредственно связана с об-

Таблица 10.1

Распределение функций компонентов приложения в моделях клиент — сервер

Модели распределений	Компоненты приложения						Пользователь
	Функции логики представления		Функции бизнес-логики		Функции управления данными		
		DP	RP	DBL	RDM	DDM	
Распределенное представление (DP)							Клиент
							Сервер
Удаленное представление (RP)							
							Клиент
							Сервер
Распределенная бизнес-логика (DBL)							
							Клиент
Удаленное управление данными (RDM)							
							Клиент
							Сервер
Распределенное управление данными (DDM)							
							Клиент
							Сервер
Совмещение DBL и DDM							
							Клиент
							Сервер

Примечание. Фоном выделено наличие соответствующих компонентов приложения.

работкой данных внутри приложения. Данными управляет собственно СУБД. Для обеспечения доступа к данным используется язык SQL.

Процессор управления данными (Database Manager System Processing) — это собственно СУБД. В идеале функции СУБД должны быть скрыты от бизнес-логики приложения, однако для рассмотрения архитектуры приложения их надо выделить в отдельную часть приложения.

В централизованной архитектуре (Host-based processing) эти части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы.

В децентрализованной архитектуре эти задачи могут быть поразному распределены между серверным и клиентским процессами. В зависимости от характера распределения можно выделить следующие модели распределений (табл. 10.1):

- распределенная презентация (DP — Distribution Presentation);
- удаленная презентация (RP — Remote Presentation);
- распределенная бизнес-логика (RBL — Remote business logic);
- распределенное управление данными (DDM — Distributed data management);
- удаленное управление данными (RDM — Remote data management).

Эта условная классификации показывает, как могут быть распределены отдельные задачи между серверным и клиентскими процессами. В этой классификации отсутствует реализация удаленной бизнес-логики. Считается, что она не может быть удалена сама по себе полностью, а может быть лишь распределена между разными процессами, которые могут взаимодействовать друг с другом.

10.3. Двухуровневые модели

Двухуровневая модель фактически является результатом распределения пяти указанных выше функций между двумя процессами, которые выполняются на двух платформах: на клиенте и на сервере. В чистом виде почти никакая модель не существует, однако рассмотрим наиболее характерные особенности каждой двухуровневой модели: модели удаленного управления данными и модели удаленного доступа к данным.

Модель удаленного управления данными. Она также называется моделью файлового сервера (FS — File Server). В этой модели презентационная логика и бизнес-логика располагаются на клиентской части. На сервере располагаются файлы с данными, и поддерживается доступ к файлам. Функции управления информационными ресурсами в этой модели находятся на клиентской части.

Рис. 10.3. Модель файлового сервера

Распределение функций в этой модели представлено на рис. 10.3.

В этой модели файлы базы данных хранятся на сервере, клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами, собственно база метаданных, находится на клиенте.

Достоинство этой модели заключается в том, что приложение разделено на два взаимодействующих процесса. При этом сервер (серверный процесс) может обслуживать множество клиентов, которые обращаются к нему с запросами. Собственно СУБД должна находиться в этой модели на клиентском компьютере.

Алгоритм выполнения клиентского запроса сводится к следующему.

1. Запрос формулируется в командах ЯМД.

2. СУБД переводит этот запрос в последовательность файловых команд.

3. Каждая файловая команда вызывает перекачку блока информации на компьютер клиента, а СУБД анализирует полученную информацию; если в полученном блоке не содержится ответ на запрос, то принимается решение о перекачке следующего блока информации, и т.д.

4. Перекачка информации с сервера на клиентский компьютер производится до тех пор, пока не будет получен ответ на запрос клиента.

Данная модель имеет следующие недостатки:

- высокий сетевой трафик, который связан с передачей по сети множества блоков и файлов, необходимых приложению;
- узкий спектр операций манипулирования с данными, который определяется только файловыми командами;
- отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы).

Модель удаленного доступа к данным. В модели удаленного доступа (RDA — Remote Data Access) база данных хранится на сервере. На сервере же находится и ядро СУБД. На компьютере клиента располагается презентационная логика и бизнес-логика при-

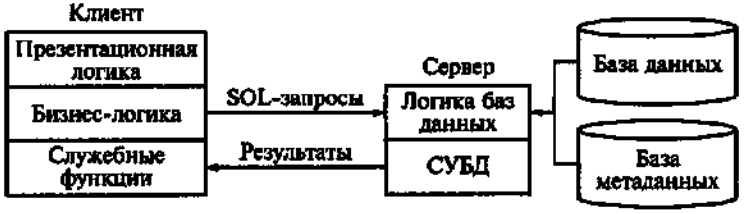


Рис. 10.4. Структура модели удаленного доступа к данным

ложения. Клиент обращается к серверу с запросами на языке SQL. Структура модели удаленного доступа приведена на рис. 10.4.

Преимущества данной модели заключаются в следующем:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгружает сервер БД, сводя к минимуму общее число выполняемых процессов в операционной системе;
- сервер БД освобождается от несвойственных ему функций; процессор или процессоры сервера целиком загружаются операциями обработки данных запросов и транзакций;
- резко уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на SQL, а их объем существенно меньше. В ответ на запросы клиент получает только данные, соответствующие запросу, а не блоки файлов.

Основное достоинство RDA-модели — унификация интерфейса клиент—сервер (стандартом при общении приложения-клиента и сервера становится язык SQL).

Данная модель имеет следующие недостатки:

- запросы на языке SQL при интенсивной работе клиентской части приложения могут существенно загрузить сеть;
- так как в этой модели на клиенте располагается и презентационная логика, и бизнес-логика приложения, то при повторении аналогичных функций в разных приложениях код соответствующей бизнес-логики должен быть повторен для каждого клиентского приложения. Это вызывает излишнее дублирование приложения;
- сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте.

10.4. Модель сервера баз данных

Для того чтобы избавиться от недостатков модели удаленного доступа, должны быть соблюдены следующие условия.

1. Необходимо, чтобы БД в каждый момент отражала текущее состояние предметной области, которое определяется не только

собственно данными, но и связями между объектами данных, т.е. данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.

2. БД должна отражать некоторые правила предметной области, законы, по которым она функционирует (business rules). Например, завод может нормально работать только в том случае, если на складе имеется некоторый достаточный запас (страховой запас) деталей определенной номенклатуры; деталь может быть запущена в производство только в том случае, если на складе имеется в наличии достаточно материала для ее изготовления, и т.д.

3. Необходим постоянный контроль за состоянием БД, отслеживание всех изменений и адекватная реакция на них. Например, при достижении некоторым измеряемым параметром критического значения должно произойти отключение определенной аппаратуры; при уменьшении товарного запаса ниже допустимой нормы должна быть сформирована заявка конкретному поставщику на поставку соответствующего товара и т.п.

4. Необходимо, чтобы возникновение некоторой ситуации в БД четко и оперативно влияло на ход выполнения прикладной задачи.

5. Одной из важнейших проблем СУБД является контроль типов данных. В настоящий момент СУБД контролирует синтаксически только стандартно-допустимые типы данных, т.е. такие, которые определены в DDL (data definition language) — языке описания данных, который является частью SQL. Однако в реальных предметных областях действуют данные, которые несут в себе еще и семантическую составляющую, например координаты объектов или единицы измерений.

Такую модель поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры. Модель активного сервера базы данных представлена на рис. 10.5.

Рис. 10.5. Модель активного сервера базы данных

В этой модели бизнес-логика разделена между клиентом и сервером. На сервере бизнес-логика реализована в виде хранимых процедур — специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД. Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все изменения в БД, которые в ней предусмотрены. Сервер возвращает клиенту данные выполненного запроса, которые требуются клиенту либо для вывода на экран, либо для выполнения части бизнес-логики. При этом трафик обмена информацией между клиентом и сервером резко уменьшается.

Централизованный контроль в модели сервера баз данных выполняется с использованием механизма триггеров. Триггеры также являются частью БД.

Термин «триггер» взят из электроники и семантически очень точно характеризует механизм отслеживания специальных событий, которые связаны с состоянием БД. Триггер в БД является неким тумблером, который срабатывает при возникновении определенного события в БД. Ядро СУБД проводит мониторинг всех событий, которые вызывают созданные и описанные триггеры в БД, и при возникновении соответствующего события сервер запускает соответствующий триггер. Каждый триггер представляет собой также некоторую программу, которая выполняется над базой данных. Триггеры могут вызывать хранимые процедуры.

Механизм использования триггеров предполагает, что при срабатывании одного триггера могут возникнуть события, которые вызовут срабатывание других триггеров.

В данной модели сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.

И хранимые процедуры, и триггеры хранятся в словаре БД. Они могут быть использованы несколькими клиентами, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.

Недостатком данной модели является очень большая загрузка сервера, так как он обслуживает множество клиентов и выполняет следующие функции:

- осуществляет мониторинг событий, связанных с описанными триггерами;
- обеспечивает автоматическое срабатывание триггеров при возникновении связанных с ними событий;
- обеспечивает исполнение внутренней программы каждого триггера;
- запускает хранимые процедуры по запросам пользователей;
- запускает хранимые процедуры из триггеров;
- возвращает требуемые данные клиенту;

• обеспечивает все функции СУБД (доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД).

Если мы перенесем на сервер большую часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшатся. Иногда такую модель называют моделью с тонким клиентом. Ранее рассмотренные модели называют моделями с толстым клиентом.

Для разгрузки сервера была предложена трехуровневая модель — модель сервера приложений.

10.5. Модель сервера приложений

Эта модель является расширением двухуровневой модели, в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Архитектура трехуровневой модели приведена на рис. 10.6. Этот промежуточный уровень содержит один или несколько серверов приложений.

В этой модели компоненты приложения делятся между тремя исполнителями: клиентом, сервером, сервером базы данных.

Клиент обеспечивает логику представления, включая графический пользовательский интерфейс, локальные редакторы; клиент может запускать локальный код приложения клиента, который может содержать обращения к локальной БД, расположенной на компьютере-клиенте. Клиент исполняет коммуникационные функции front-end части приложения, которые обеспечивают доступ клиенту в локальную или глобальную сеть. Дополнительно реализация взаимодействия между клиентом и сервером может включать в себя управление распределенными транзакциями, что соответствует тем случаям, когда клиент также является клиентом менеджера распределенных транзакций.

Серверы приложений составляют новый промежуточный уровень архитектуры. Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, поддерживают сетевую доменную операционную среду, хранят и исполняют наиболее общие правила бизнес-логики, поддерживают каталоги с данными, обеспечивают обмен сообщениями и поддержку запросов, особенно в распределенных транзакциях.

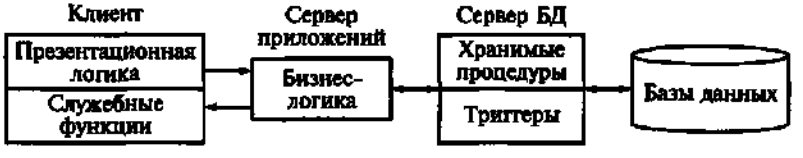


Рис. 10.6. Модель сервера приложений

Серверы баз данных в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных (warehouse services). Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших (унаследованных) приложений (legacy application).

Эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных, которые относятся к области OLAP-приложений (On-line analytical processing). В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного SQL, реализованного в конкретной СУБД, и может быть выполнена на языках программирования, таких как C, C++, Cobol. Это повышает переносимость системы, ее масштабируемость.

10.6. Модели серверов баз данных

В период создания первых СУБД технология клиент — сервер только зарождалась. Поэтому изначально в архитектуре систем не было адекватного механизма организации взаимодействия процессов типа «клиент» и процессов типа «сервер». В современных же СУБД он является фактически основополагающим и от эффективности его реализации зависит эффективность работы системы в целом.

Рассмотрим эволюцию типов организации подобных механизмов. В основном этот механизм определяется структурой реализации серверных процессов, и часто он называется *архитектурой сервера баз данных.*

Первоначально, как мы уже отмечали, существовала модель, у которой управление данными (функция сервера) и взаимодействие с пользователем были совмещены в одной программе. Это можно назвать нулевым этапом развития серверов БД.

Затем функции управления данными были выделены в самостоятельную группу — сервер, однако модель взаимодействия пользователя с сервером соответствовала структуре связей между таблицами баз данных «один к одному» (рис. 10.7), т.е. сервер обслуживал запросы только одного пользователя (клиента), а для обслуживания нескольких клиентов нужно было запустить эквивалентное число серверов.

Выделение сервера в отдельную программу было революционным шагом, который позволил, в частности, поместить сервер на одну машину, а программный интерфейс с пользователем — на другую, осуществляя взаимодействие между ними по сети. Одна-

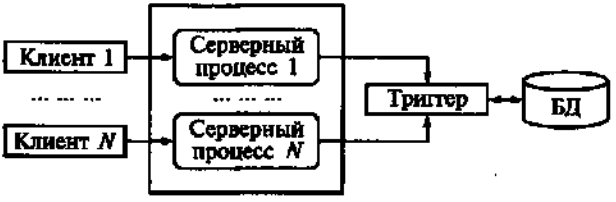


Рис. 10.7. Взаимодействие клиентских и серверных процессов в модели «один к одному»

ко необходимость запуска большого числа серверов для обслуживания множества пользователей сильно ограничивала возможности такой системы.

Для обслуживания большого числа клиентов на сервере должно было быть запущено большое число одновременно работающих серверных процессов, а это резко повышало требования к ресурсам ЭВМ.

Кроме того, каждый серверный процесс в этой модели запускался как независимый, поэтому если один клиент сформировал запрос, который был только что выполнен другим серверным процессом для другого клиента, то запрос выполнялся повторно. В такой модели весьма сложно обеспечить взаимодействие серверных процессов. Эта модель самая простая, и она появилась первой.

Проблемы, возникающие в информационной модели «один к одному», решаются в архитектуре систем с выделенным сервером, который способен обрабатывать запросы от многих клиентов. Сервер единственный обладает монополией на управление данными и взаимодействует одновременно со многими клиентами (рис. 10.8). Логически каждый клиент связан с сервером отдельной нитью (thread), или потоком, по которому пересылаются запросы. Такая архитектура получила название *многопотоковой односерверной* (multi-threaded). Она позволяет значительно уменьшить нагрузку на операционную систему, возникающую при работе большого числа пользователей (trashing).

Кроме того, возможность взаимодействия многих клиентов с одним сервером позволяет в полной мере использовать разделя-

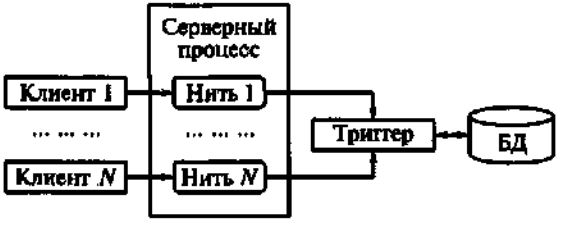


Рис. 10.8. Многопотоковая односерверная архитектура

емые объекты (начиная с открытых файлов и кончая данными из системных каталогов), что значительно уменьшает потребности в памяти и общее число процессов операционной системы. Например, системой с моделью «один к одному» будет создано 100 копий процессов СУБД для 100 пользователей, тогда как системе с многопотоковой архитектурой для этого понадобится только один серверный процесс.

Однако такое решение имеет свои недостатки. Так как серверный процесс может выполняться только на одном процессоре, возникает естественное ограничение на применение СУБД для мультипроцессорных платформ. Если компьютер имеет, например, четыре процессора, то СУБД с одним сервером используют только один из них, не загружая оставшиеся три.

В некоторых системах эта проблема решается вводом промежуточного диспетчера. Подобная архитектура называется *архитектурой виртуального сервера* (virtual server) (рис. 10.9).

В этой архитектуре клиенты подключаются не к реальному серверу, а к промежуточному звену, называемому диспетчером, который выполняет только функции диспетчеризации запросов к серверам. В этом случае нет ограничений на использование многопроцессорных платформ. Число серверов может быть согласовано с числом процессоров в системе.

Однако и эта архитектура не лишена недостатков, потому что здесь в систему добавляется новый слой, который размещается между клиентом и сервером, что увеличивает потребность в ресурсах на поддержку баланса загрузки серверов (load balancing) и ограничивает возможности управления взаимодействием клиент—сервер. Во-первых, становится невозможным направить запрос от конкретного клиента конкретному серверу; во-вторых, серверы становятся равноправными, так как нет возможности устанавливать приоритеты для обслуживания запросов.

Подобная организация взаимодействия между клиентом и сервером может рассматриваться как аналог банка, где имеется несколько окон кассиров и специальный банковский служащий, администратор зала, который направляет каждого пришедшего посетителя (клиента) к свободному кассиру (актуальному серверу).

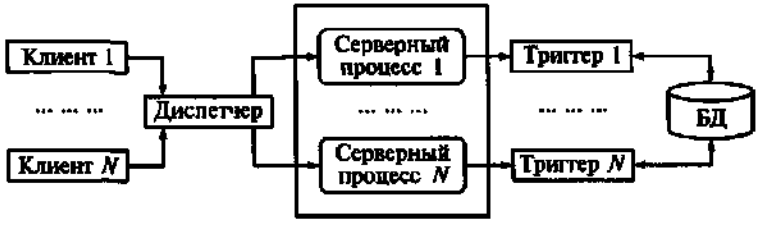


Рис. 10.9. Архитектура виртуального сервера

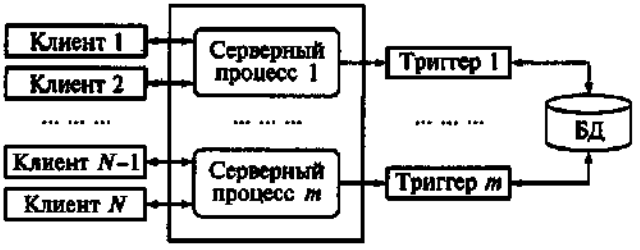


Рис. 10.10. Многопотоковая мультисерверная архитектура

Система работает нормально, пока все посетители равноправны (имеют равные приоритеты), однако стоит появиться посетителям с высшим приоритетом, которые должны обслуживаться в специальном окне, как возникают проблемы. Учет приоритета клиентов особенно важен в системах оперативной обработки транзакций, однако именно эту возможность не может предоставить архитектура систем с диспетчеризацией.

Современное решение проблемы СУБД для мультипроцессорных платформ заключается в возможности запуска нескольких серверов базы данных, в том числе и на различных процессорах. При этом каждый из серверов должен быть многопотоковым. Если эти два условия выполнены, то есть основания говорить о многопотоковой архитектуре с несколькими серверами, представленной на рис. 10.10.

Такую архитектуру называют также *многонитиевой мультисерверной архитектурой*. Эта архитектура обеспечивает распараллеливание выполнения одного пользовательского запроса несколькими серверными процессами.

В этом случае пользовательский запрос разбивается на ряд подзапросов, которые могут выполняться параллельно, а результаты их выполнения потом объединяются в общий результат выполнения запроса. Тогда для обеспечения оперативности выполнения

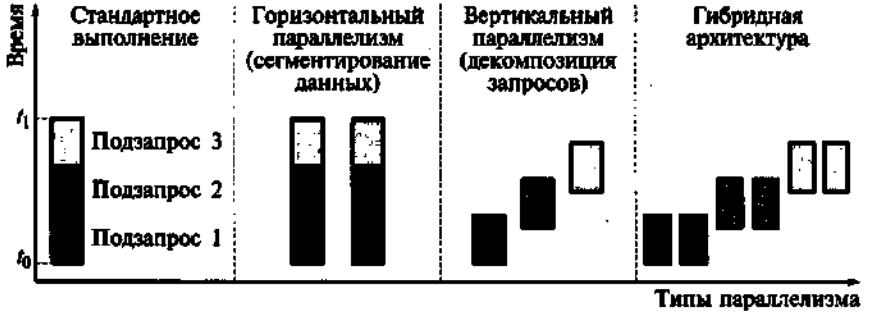


Рис. 10.11. Многонитевая мультисерверная архитектура

запросов их подзапросы могут быть направлены отдельным серверным процессам, а затем полученные результаты объединены в общий результат (рис. 10.11). В данном случае серверные процессы не являются независимыми процессами — такими, как рассматривались ранее. Эти серверные процессы принято называть *нитями* (treads). Управление нитями множества запросов пользователей требует дополнительных расходов от СУБД, однако при оперативной обработке информации в хранилищах данных такой подход наиболее перспективен.

10.7. Типы параллелизма

Рассматривают следующие программно-аппаратные способы распараллеливания запросов: горизонтальный, вертикальный и смешанный параллелизм.

Горизонтальный параллелизм. Этот параллелизм возникает тогда, когда хранимая в БД информация распределяется по нескольким физическим устройствам хранения — нескольким дискам. При этом информация из одного отношения разбивается на части по горизонтали. Этот вид параллелизма иногда называют распараллеливанием, или сегментацией, данных. Параллельность достигается путем выполнения одинаковых операций, например фильтрации, над разными физическими хранимыми данными. Эти операции могут выполняться параллельно разными процессами — они независимы. Результат выполнения целого запроса складывается из результатов выполнения отдельных операций.

Время выполнения такого запроса при соответствующем сегментировании данных существенно меньше, чем время выполнения этого же запроса традиционными способами одним процессом.

Вертикальный параллелизм. Этот параллелизм достигается конвейерным выполнением операций, составляющих запрос пользователя. Этот подход требует серьезного усложнения модели выполнения реляционных операций ядром СУБД. Он предполагает, что ядро СУБД может произвести декомпозицию запроса, базируясь на его функциональных компонентах; при этом ряд подзапросов может выполняться параллельно, с минимальной связью между отдельными шагами выполнения запроса.

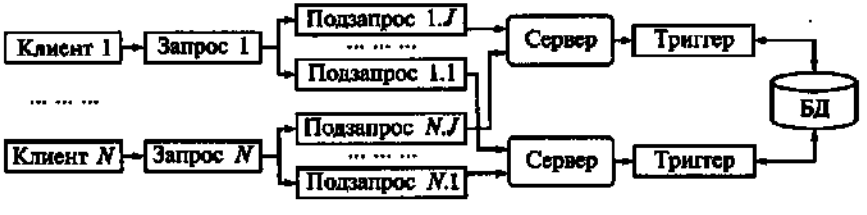


Рис. 10.12. Схема выполнения запроса при вертикальном параллелизме

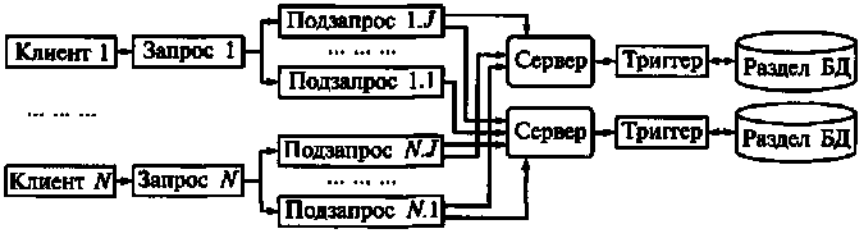


Рис. 10.13. Схема выполнения запроса при смешанном параллелизме

Рассмотрим, например, следующую последовательность операций реляционной алгебры:

1. R5 = R1 [A,C].
2. R6 =R2 [A,B,D].
3. R7 = R5[A > 128].
4. R8 = R5[A]R6.

Тогда первую и третью операции можно объединить и выполнить параллельно со второй операцией, а затем выполнить над результатами последнюю — четвертую — операцию.

Общее время выполнения подобного запроса, конечно, будет существенно меньше, чем при традиционном способе выполнения последовательности из четырех операций (рис. 10.12).

Смешанный параллелизм. Этот параллелизм является гибридом горизонтального и вертикального параллелизма (рис. 10.13).

Все виды параллелизма применяются в приложениях, где они позволяют существенно сократить время выполнения сложных запросов над очень большими объемами данных.

Контрольные вопросы

1. Дайте определения следующих понятий:
   - топология БД, или структура распределенной БД;
   - локальная автономность;
   - удаленный запрос;
   - поддержка распределенной транзакции;
   - презентационная логика;
   - бизнес-логика.
2. Какие двухуровневые модели вы знаете? Назовите их достоинства и недостатки.
3. Назовите характеристики следующих архитектур организации баз данных:
   - многопотоковая односерверная архитектура;
   - архитектура с виртуальным сервером;
   - многонитиевая мультисерверная архитектура.
4. Для чего применяют распараллеливание запросов и какие типы параллелизма вы знаете?

Глава 11

СЕТЕВАЯ БАЗА ДАННЫХ SQL SERVER 2000

11.1. Компоненты SQL Server 2000

SQL Server 2000 является одной из современных систем управления базами данных в масштабах крупных предприятий.

Для профессиональной работы с SQL Server 2000 необходимо понимать принципы его функционирования, знать, какой из компонентов необходимо использовать в том или ином случае. Рассмотрим компоненты (службы) SQL Server 2000, их назначение и методы использования.

Как и многие серверные продукты, работающие под управлением операционной системы Windows NT или Windows 2000, Microsoft SQL Server 2000 реализован в виде набора служб операционной системы, каждая из которых запускается самостоятельно и отвечает за определенный круг задач.

Приведем список служб SQL Server:

- MSSQLServer;
- SQLServerAgent;
- Microsoft Search;
- Microsoft Distributed Transaction Coordinator.

Реализация SQL Server 2000 в виде отдельных служб позволяет СУБД работать как части операционной системы, иметь собственные права доступа и не зависеть от пользователя, работающего на компьютере в данный момент. Операционная система Windows 95/98 не поддерживает работу этих служб, поэтому для работы SQL Server 2000 под управлением этой операционной системы автоматически выполняется эмуляция служб. Рассмотрим более подробно каждую из служб SQL Server 2000.

Служба MSSQLServer. Служба MSSQLServer является ядром СУБД и выполняет все основные операции. В задачи службы MSSQLServer входит регистрация пользователей, контроль их прав доступа, установление соединения, обслуживание обращений пользователей к базам данных, выполнение хранимых процедур, работа с файлами баз данных и многое другое.

В функции службы MSSQLServer входит также контроль за использованием системных ресурсов. Служба MSSQLServer периодически опрашивает систему о наличии свободных ресурсов и при достаточном их числе автоматически выделяет дополнительную память или процессорное время. Полученные ресурсы распределяются между всеми подключенными пользователями, тем

самым достигается максимальная производительность обработки запросов. При использовании многопроцессорной системы выполняется распараллеливание «тяжелых» запросов пользователей между всеми доступными процессорами.

Все остальные службы можно рассматривать как расширения службы MSSQLServer, добавляющие гибкость и функциональность СУБД. Служба MSSQLServer всегда запускается первой. Только после ее успешного старта могут быть запущены и начать свою работу другие службы.

Служба SQLServerAgent. Служба SQLServerAgent прежде всего предназначена для автоматизации администрирования СУБД. В задачи этой службы входит автоматический запуск заданий и извещение операторов о сбоях в работе сервера. С помощью службы SQLServerAgent можно выполнять запуск различных задач в определенное время, что может избавить администратора от большей части рутинной работы. Например, администратор может спланировать автоматическое выполнение операций резервного копирования и проверки целостности информации в базе данных во время наименьшей активности пользователей. При этом администратору не нужно будет контролировать ход выполнения операций.

Большая часть операций, выполняемых службой SQLServerAgent, реализована в виде системных хранимых процедур, которые выполняются службой MSSQLServer. В работе службы SQLServerAgent применяются объекты трех типов:

- Jobs (задания);
- Operators (операторы);
- Alerts (события).

Информация обо всех этих объектах, включая расписание автоматического запуска задач, хранится в системной базе данных Msdb. При каждом старте SQLServerAgent анализирует содержание этой базы данных. Если к моменту запуска службы накопились просроченные задания или произошло сконфигурированное событие, то служба выполняет соответствующие действия.

Для управления заданиями, операторами и событиями можно пользоваться различными методами. Наиболее удобно использовать графический интерфейс утилиты Enterprise Manager. Второй способ заключается в вызове системных хранимых процедур и команд Transact-SQL. Третий способ предполагает обращение к интерфейсу SQL-DMO. В последнем случае возможно написание своих собственных приложений, обеспечивающих интерфейс работы с заданиями, операторами и событиями.

Квалифицированный подход к работе службы SQLServerAgent может снизить расходы на сопровождение баз данных, в частности за счет уменьшения числа операторов и администраторов. Экономический эффект от применения службы SQLServerAgent про-

порциональна размеру предприятия: чем больше предприятие, тем больше пользы оно получит от всех возможностей SQL Server 2000.

Объекты Jobs. Объекты этого типа описывают задачи, которые должны быть выполнены автоматически. Для каждого задания указывается одно или более расписаний его запуска (schedule). Кроме того, задание может быть выполнено по требованию (on demand), т.е. вручную. Каждое задание состоит из одного или более шагов (step). В качестве шага могут выступать:

- команда или запрос Transact-SQL;
- команды управления подсистемой репликации;
- утилиты командной строки или приложения Windows;
- скрипты, написанные на языках VB Script или JavaScript, и др.

Служба SQLServerAgent обладает возможностью контроля правильности выполнения заданий, позволяет создавать многошаговые задания. Шаги могут быть связаны между собой по определенным правилам. Например, если проверка целостности базы данных завершилась успешно, то служба создаст резервную копию данных; в противном случае сервер может отправить соответствующее извещение администратору по электронной почте или на пейджер. Служба позволяет гибко управлять временем запуска задач, обеспечивая их выполнение как в определенное время, так и в моменты наименьшей загруженности сервера.

Объекты Operators. Объекты этого типа описывают операторов — служащих, отвечающих за поддержание сервера в рабочем состоянии. В небольших организациях роли оператора и администратора обычно совмещает один человек. На больших предприятиях и в корпорациях роли администратора и оператора чаще всего разделены между несколькими людьми. Администратор выполняет только ответственную работу, например планирование, создание и изменение баз данных. Оператор же чаще занимается рутинной работой, такой как выполнение резервного копирования базы данных, добавление пользователей, контроль за целостностью данных и т.д. Если организация большая, то можно использовать специализированных операторов. Например, один из операторов будет ответственен за выполнение операций резервного копирования, другой станет следить за целостностью данных и т.д. Каждый из операторов должен получать сообщения, относящиеся к его виду деятельности. Нежелательно, чтобы оператор резервного копирования начал разрешать проблемы мертвых блокировок.

SQL Server 2000 отслеживает параметры своей работы и при обнаружении неполадок, например при недостатке свободного пространства на диске, может известить оператора о неприятностях. Для этого используется служба SQLServerAgent. Необходимо предварительно сконфигурировать операторов и указать события, при наступлении которых будет отправляться извещение операто-

рам. Служба SQLServerAgent для извещения операторов может рассылать определенные сообщения по электронной почте или отправлять сообщения непосредственно на пейджер оператора. Кроме того, для извещения оператора допускается вызов команды *net send*, с помощью которой можно отправить сообщение по локальной сети. Можно настроить запуск команды таким образом, что сообщение получат все пользователи сети, в надежде на то, что кто-нибудь из них сообщит оператору о проблемах в работе сервера. Но чаще всего команда *net send* служит для посылки сообщения конкретному пользователю.

Объекты Alerts. Объекты типа Alerts описывают события, на которые должен реагировать SQL Server 2000. При наступлении описанного события сервер с помощью службы SQLServerAgent отправляет одному или нескольким операторам извещение об обнаружении неполадок в работе сервера. События SQL Server 2000 охватывают почти все аспекты работы сервера, что позволяет эффективно контролировать работу SQL Server 2000. Операторам не обязательно постоянно находиться рядом с сервером, чтобы знать о параметрах его работы. Оператор может даже не присутствовать в здании при обнаружении сбоя, но он может получить извещение на пейджер и предпринять необходимые действия, в том числе и удаленно. Описанный подход позволяет снизить затраты на сопровождение баз данных в больших организациях. Отпадает необходимость в персональном операторе для каждого из серверов предприятия.

Служба Microsoft Search (MSSearch). Служба Microsoft Search, также называемая Full-Text Search, используется для поиска символьной информации в таблицах баз данных. Служба Microsoft Search позволяет выполнять полнотекстовый поиск. Технология полнотекстового поиска позволяет находить не только конкретные слова и фразы, но и близкие к ним по смыслу и написанию. Для реализации полнотекстового поиска существуют полнотекстовые каталоги (full-text catalog) и полнотекстовые индексы (full-text index). Данные полнотекстовых каталогов и индексов хранятся отдельно от основных данных в специальных файлах. Все действия по работе с этими файлами осуществляет служба MSSearch. Связь между службами MSSQLServer и MSSearch осуществляется через специального поставщика (full-text provider).

Служба MSSearch периодически анализирует содержание таблиц баз данных и обновляет (repopulation) полнотекстовые каталоги и индексы. Если необходимо создать полнотекстовый индекс заново, то следует выполнить перестроение (rebuild) индекса. Результатом такого подхода является то, что данными полнотекстового поиска нужно управлять отдельно от основных данных. Администратор должен настроить интервалы обновления данных полнотекстового поиска. Кроме того, операции резервного копирования

и восстановления файлов полнотекстового поиска необходимо выполнять отдельно от основных данных.

Служба Microsoft Distributed Transaction Coordinator (MSDTC). SQL Server 2000 дает возможность пользователям работать одновременно с несколькими источниками данных. Пользователи в одном запросе могут обращаться к различным базам данных, хранящимся на одном и том же или на разных серверах. Кроме того, пользователи могут обращаться не только к серверам SQL Server 2000, но и к любым источникам данных, работающим с технологией OLE DB. Эта технология позволяет обращаться не только к реляционным источникам данных, таким как Oracle, FoxPro, MS Access, но и к нереляционным источникам данных, таким как текстовые файлы, книги MS Excel, и к другим приложениям.

Для обращения из тела одной транзакции к множеству источников данных SQL Server 2000 использует распределенные транзакции (distributed transaction). Для управления распределенными транзакциями существует координатор распределенных транзакций (Distributed Transaction Coordinator). В SQL Server 2000 координатор распределенных транзакций реализован в виде службы MSDTC. Эта служба автоматически отслеживает ситуации, в которых необходимо начать выполнение распределенных транзакций. В некоторых ситуациях пользователь может и не подозревать, что его транзакция выполняется как распределенная. Служба MSDTC скрывает от пользователя все действия по обработке распределенных транзакций. Распределенные транзакции реализуются как множество локальных транзакций, открываемых на каждом источнике данных координатором распределенных транзакций. Служба MSDTC синхронизирует все транзакции таким образом, что обеспечивается целостность данных на всех участниках распределенной транзакции. Это достигается использованием специального двухфазного протокола изменений (2PC, two-phase commit protocol).

11.2. Системные базы данных SQL Server 2000

SQL Server 2000 в своей работе использует несколько системных баз данных. Эти базы данных создаются автоматически при установке SQL Server 2000 и не должны удаляться. Вся информация о настройке сервера хранится в этих базах данных. Их можно сравнить с реестром операционной системы Windows, в котором хранится вся системная и пользовательская информация. Удаление или повреждение реестра приведет к краху системы и невозможности ее работы. Аналогичная ситуация наблюдается и с системными базами данных SQL Server 2000. Приведем список системных баз данных:

- Master;
- Model;

• Tempdb;
• Msdb.

При работе с реестром операционной системы пользователи чаще всего выбирают специальные инструменты, например утилиты панели управления. Не рекомендуется напрямую работать с реестром, так как даже небольшие изменения могут существенно повредить работоспособности системы. Таким же образом следует поступать и при работе с системными базами данных. Не рекомендуется напрямую работать с ними с помощью команд *Select*, *Insert*, *Update*, *Delete*. Пользователи могут только считывать данные с помощью команды *Select*. Для изменения данных в системных таблицах в SQL Server 2000 имеется набор системных хранимых процедур, с помощью которых можно выполнить практически любые действия по администрированию сервера.

База данных Master. Эта системная база данных является главной базой данных SQL Server 2000. Она выполняет функции реестра операционной системы Windows. Остальные системные базы данных имеют второстепенное значение и их можно считать вспомогательными. В базе данных Master хранится вся системная информация о параметрах конфигурации сервера, имеющихся на сервере пользовательских баз данных, пользователях, имеющих доступ к серверу, и другая системная информация.

По умолчанию база данных Master создается в каталоге Data установочного каталога SQL Server 2000. База данных состоит из двух файлов:

• Master.mdf — основной файл базы данных, содержащий собственно данные. Размер этого файла после установки составляет 8 Мбайт;

• Master.ldf — файл базы данных, предназначенный для хранения журнала транзакций. Размер этого файла после установки составляет 1 Мбайт.

База данных Model. Эта системная база данных является шаблоном для создания новых баз данных. Технология создания новой базы данных в SQL Server 2000 построена так, что сервер копирует базу данных Model в указанное место и соответствующим образом изменяет ее имя. Если при создании базы данных не указаны никакие параметры, кроме ее имени, то новая база данных будет являться полной копией базы данных Model. Если же размер и состав файлов создаваемой базы данных указан явно, то скопированная база данных изменяется соответствующим образом. Но в любом случае в качестве основы используется база данных Model.

Изменяя параметры базы данных Model, можно управлять параметрами создаваемых по умолчанию баз данных. Кроме того, базу данных Model можно использовать в качестве корпоративного стандарта на содержимое и свойства базы данных, т.е. админи-

стратор может создать в базе данных Model набор таблиц и хранимых процедур, которые должны быть в каждой базе данных.

После установки SQL Server 2000 размер базы данных Model установлен в 1,5 Мбайт. База данных Model располагается в каталоге Data и состоит из двух файлов размером по 0,75 Мбайт каждый:

- Model.mdf — основной файл базы данных, содержащий собственно данные;
- Model.ldf — файл базы данных, используемый для хранения журнала транзакций.

База данных Tempdb. База данных Tempdb (полное название — Temporary DataBase) предназначена для хранения всех временных объектов, создаваемых пользователями во время сеанса работы. Если постоянные объекты, такие как таблицы или представления, создаются в пользовательской базе данных, то временные объекты возникают в базе данных Tempdb. Доступ к базе данных Tempdb автоматически имеется у всех пользователей, и администратор не должен предпринимать никаких действий для предоставления им доступа к этой базе данных.

Отличительной особенностью базы данных Tempdb является то, что она уничтожается каждый раз, когда происходит останов сервера. Естественно, все временные объекты, созданные пользователями, также уничтожаются. При следующем запуске SQL Server 2000 база данных Tempdb создается заново. При создании базы данных Tempdb, также как и для пользовательских баз данных, в качестве основы применяется база данных Model, при этом наследуются все ее свойства. Администратор должен учитывать это, изменяя параметры базы данных Model. Неверное конфигурирование параметров этой базы данных может неблагоприятным образом повлиять на работу всех пользователей. Кроме того, при планировании параметров базы данных Tempdb следует учитывать требования к свободному пространству на диске. Как и для всех баз данных, для Tempdb поддерживается возможность автоматического роста файлов базы данных. При интенсивном обращении пользователей к ресурсам базы данных Tempdb неизбежен ее рост. Необходимо правильно выбрать первоначальный размер и шаг прироста этой базы данных. Неверное конфигурирование этих параметров может заметно снизить производительность системы.

База данных Tempdb состоит из двух файлов, располагающихся в каталоге Data установочного каталога SQL Server 2000:

- Tempdb.mdf — основной файл базы данных, содержащий временные объекты. Размер этого файла после установки составляет 8 Мбайт;
- Tempdb.ldf — журнал транзакций. Размер этого файла после установки составляет 0,5 Мбайт.

База данных Msdb. Системная база данных Msdb предназначена для размещения информации, используемой службой

SQLServerAgent, относящейся к автоматизации администрирования и управлений SQL Server 2000, а также информации об операторах и событиях.

11.3. Инструменты SQL Server 2000

Обычно инструменты администрирования устанавливаются при инсталляции SQL Server 2000. Тем не менее они могут быть добавлены и отдельно. Рассмотрим инструменты SQL Server 2000.

Enterprise Manager. Данный инструмент является базовым для выполнении самых разнообразных задач:

- управления системой безопасности;
- создания баз данных и ее объектов;
- создания и восстановление резервных копий;
- конфигурирования подсистемы репликации;
- управления параметрами работы служб SQL Server 2000;
- управления подсистемой автоматизации;
- запуска и останова работы служб;
- конфигурирования связанных и удаленных серверов;
- создания, управления и выполнения пакетов DTS.

Приведенный список не исчерпывает всех областей применения Enterprise Manager и может быть расширен. Однако перечисленных задач достаточно, чтобы понять всю важность этого инструмента.

Большая часть административных задач SQL Server 2000 может быть выполнена следующими методами:

- использование средств Transact-SQL;
- с помощью графического интерфейса Enterprise Manager;
- с помощью мастеров (wizards).

Порядок перечисления методов соответствует уменьшению сложности работы с ними. Самым сложным является выполнение задачи средствами Transact-SQL, так как это требует знания синтаксиса команд и хранимых процедур. Использование средств Transact-SQL обеспечивает пользователю прямой доступ к системным данным.

SQL Server Service Manager. Единственной задачей утилиты SQL Server Service Manager является предоставление пользователю удобного механизма запуска, останова и приостановка служб SQL Server 2000. Кроме того, она позволяет лишь запретить или разрешить автоматический запуск той или иной службы при загрузке операционной системы.

Утилита Service Manager устанавливается при инсталляции SQL Server 2000 и по умолчанию автоматически запускается при загрузке операционной системы.

SQL Server Profiler. Утилита SQL Server Profiler — это графический инструмент, с помощью которого администратор может

наблюдать за теми или иными аспектами работы SQL Server 2000. В основе работы этой утилиты лежит тот же принцип, что и в основе работы утилиты Performance. При выполнении пользовательских запросов, хранимых процедур, команд Transact-SQL, подключении к серверу и отключении от него, а также множества других действий ядро SQL Server 2000 сохраняет в системных таблицах массу различной информации о ходе выполнения операций. Эта информация может быть получена с помощью специальных хранимых процедур. Утилита SQL Server Profiler использует эти хранимые процедуры для получения необходимой информации. Полученные данные затем предоставляются в удобном виде с помощью графического интерфейса. Однако пользователи могут получать информацию о процессах SQL Server 2000, обращаясь напрямую к хранимым процедурам. На основе этих хранимых процедур можно даже написать свое собственное приложение, которое будет отображать информацию о работе SQL Server 2000 в нужной форме.

Мониторинг работы SQL Server 2000 основывается на наблюдении за событиями (events). Событие генерируется ядром SQL Server 2000 и является минимальным объемом работы, который можно контролировать. Каждое событие принадлежит к какому-то классу событий (event classes), который описывает его параметры и смысл той или иной информации. Для лучшего понимания разницы между событием и классами событий SQL Server Profiler проведем аналогию с объектами и экземплярами объектов Performance Monitor. Класс событий SQL Server Profiler, как и объект Performance Monitor, представляет собой абстрактное описание, тогда как само событие (экземпляр объекта) представляет собой информацию о работе того или иного объекта.

Число классов событий SQL Server довольно велико. Для облегчения работы с ними они были разбиты на 12 категорий (category).

Query Analyzer. Этот инструмент предназначен для выполнения запросов и анализа их исполнения. По частоте использования и важности Query Analyzer сравним с Enterprise Manager.

Одной из особенностей является возможность трассировки выполнения хранимых процедур. При выполнении трассировки пользователи могут использовать точки останова (break points), а также осуществлять пошаговое выполнение команд процедуры.

Помимо выполнения запросов и хранимых процедур с помощью Query Analyzer можно оценить производительность исполнения запроса. Для этого следует разрешить отображение оценочного (estimated) или результирующего плана исполнения (execution plan) запроса, что можно выполнить с помощью меню *Query*, выбрать в нем соответственно пункты *Display Estimated Execution Plan* и *Display Execution Plan*.

Оценочный план исполнения запроса формируется на основе предположений сервера о затратах на выполнение отдельных шагов запроса.

Результирующий же план исполнения запроса генерируется после выполнения запроса и отражает реальное положение дел. Конечно, в идеальной ситуации значения оценочного и результирующего планов исполнения будут совпадать. Однако при работе с многопользовательскими системами вполне может оказаться, что реально исполнение запроса займет больше времени, чем ожидалось.

Чаще всего это происходит из-за занятости процессора выполнением запросов других пользователей или блокированием необходимых для выполнения запроса ресурсов другими транзакциями.

Upgrade Wizard. Мастер Upgrade Wizard предназначен для выполнения обновления баз данных SQL Server 6.5 до SQL Server 2000.

В процессе обновления на SQL Server 2000 будут перенесены собственно данные, а также весь набор объектов обновляемой базы данных, включая хранимые процедуры, триггеры, правила, умолчания, ограничение целостности, представления. Кроме того, также окажутся перенесенными пользователи базы данных со всеми установленными правами доступа к объектам базы данных и т.д. Кроме того, в процессе обновления также будут скопированы все настройки подсистемы репликации.

Import and Export Data. Этот инструмент является ничем иным, как мастером импорта (экспорта) данных, предназначенным для создания пакета DTS, который будет выполнять копирование информации между двумя источниками данных. Отличительной особенностью мастера является простота конфигурирования процесса копирования данных.

К недостаткам использования мастера относится невозможность обработки более двух источников данных, а также определения сложных преобразований и отношений предшествования. Кроме того, большая часть возможностей DTS, например отправка сообщений по электронной почте, будет недоступна. Несомненным достоинством использования мастера является легкость решения простых задач. Если необходимо закачать в таблицу базы данных информацию из файла MS Excel, то возможностей мастера будет вполне достаточно. Таким образом, даже неопытные пользователи смогут выполнять основные операции обмена данными.

Client Network Utility и Server Network Utility. Наличие сетевого протокола не достаточно для сетевой работы SQL Server 2000. Для того чтобы клиенты смогли установить соединение с сервером, как на клиенте, так и на и сервере необходимо добавить специальные сетевые библиотеки (Network Library). Эти библиотеки ре-

ализованы в виде динамически подключаемых библиотек (DLL, dynamic link library) и подключаются к операционной системе. Библиотека расширяет базовые возможности протокола и является как бы надстройкой над ним, выполняющей различные сетевые операции по обмену данными между клиентом и сервером, для чего используются механизмы IPC.

Библиотеки можно инсталлировать как в процессе установки SQL Server 2000, так и позднее. Если требуется добавить или удалить библиотеку уже после установки, то для этого нужно использовать утилиту Server Network Utility, устанавливающуюся вместе с SQL Server 2000.

С помощью этой библиотеки конфигурируются сетевые параметры собственно сервера, т.е. указываются сетевые библиотеки, с помощью которых пользователи смогут обращаться к серверу. Однако со стороны клиента также требуется присутствие сетевых библиотек и конфигурирование их для работы с сервером. Конфигурирование клиента выполняется с помощью утилиты Client Network Utility, добавляемой при установке инструментов администрирования SQL Server.

Сконфигурированные параметры будут использоваться для работы Enterprise Manager, Query Analyzer и других инструментов администрирования. Чтобы гарантировать, что взаимодействие клиента с сервером окажется успешным, следует обеспечить использование клиентом хотя бы одной библиотеки, поддержка которой разрешена на сервере, а также при необходимости соответствующим образом указать ее свойства.

Утилиты командной строки. Помимо уже рассмотренных утилит, имеющих графический интерфейс, в SQL Server 2000 существует набор утилит командной строки, с помощью которых также можно выполнять различные задачи. Некоторые из этих утилит используются сервером автоматически и являются скорее частью ядра SQL Server 2000, чем собственно утилитами.

Мастера. Как уже было сказано ранее, многие задачи могут быть выполнены с помощью средств Transact-SQL, с использованием графического интерфейса Enterprise Manager и посредством специальных мастеров (wizards). Мастера являются наиболее простым способом выполнения административных задач. Недостатком мастеров являются достаточно ограниченные возможности.

Однако к некоторым из мастеров сказанное не относится. Мастера конфигурирования подсистемы репликации являются довольно сложным процессом. Например, создать публикацию средствами Enterprise Manager нельзя. Нужно будет воспользоваться соответствующим мастером. Конечно, всегда можно воспользоваться средствами Transact-SQL. Но иногда это настолько сложно и трудоемко, что лучшим решением будет использование мастера.

Контрольные вопросы

1. Назовите основное назначение следующих служб SQL Server:
   - MSSQLServer;
   - SOLServerAgent;
   - Microsoft Search (MSSearch);
   - Microsoft Distributed Transaction Coordinator (MSDTC).
2. Назовите основное назначение следующих системных баз данных SQL Server:
   - Master;
   - Model;
   - Tempdb;
   - Msdb.
3. Какие Инструменты SQL Server 2000 вы знаете?

Глава 12

СИСТЕМА УПРАВЛЕНИЯ РАСПРЕДЕЛЕННЫМИ БАЗАМИ ДАННЫХ ORACLE

12.1. Краткая история создания

В 1977 г. Л. Эллисон, Б. Майнер и Э. Оуэтс организовали фирму Relational Software Incorporated (RSI), которая и положила начало системе управления распределенными базами данных (СУРБД) Oracle. Б. Майнер и Д. Оуэтс решили разработать систему управления распределенными базами данных, используя языки C и SQL. В 1979 г. на рынок программных систем была представлена СУРБД Oracle 2, которая работала под управлением операционной системы ОС RSX-11.

В 1983 г. появилась новая версия системы Oracle 3. Разработчики внесли изменения в язык SQL, что позволило увеличить производительность системы. В отличие от Oracle 2 третья версия была полностью написана на языке C. С этого момента фирма RSI сменила свое название на Oracle Corporation.

Фирма постоянно совершенствует свою СУРБД в направлении повышения «интеллектуализации» автоматизированных систем и, что очень важно, расширения возможностей для аналитической обработки данных.

12.2. Основные понятия и терминология

Рассмотрим термины и понятия системы Oracle, отличающиеся от уже изложенных.

Блок (*Block*) — самая маленькая единица хранения данных в СУРБД Oracle. Содержит заголовочную информацию и сам блок (данные или PL/SQL-код). Размер блока составляет от 2 до 16 Кбайт.

Буфер — это некоторый объем оперативной памяти, используемый для хранения данных. Буфер содержит данные, которые предполагается использовать или которые использовались совсем недавно. В большинстве случаев буфер содержит копию данных, которые хранятся на жестком диске. Данные в буфере могут быть изменены и записаны на диск или могут быть помещены в буфер для временного хранения. Применительно к Oracle буферы содержат те блоки данных, к которым недавно обращались. Совокупность буферов составляет *кэш буферов данных*. Также в буфере сохраняются временные записи журнала изменений, которые затем записываются на диск (буфер журнала изменений).

Грязный буфер (*dirty buffer*) — буфер, содержимое которого изменилось. DBWR периодически сбрасывает грязные буферы на жесткий диск.

Динамические таблицы характеристик (*Dynamic Performance Tables*) — это таблицы, которые автоматически создаются при запуске экземпляра Oracle и используются для хранения характеристик этого экземпляра. Они включают в себя информацию о соединениях, вводе (выводе), первоначальные значения параметров среды и др.

Запрос — это транзакция «только для чтения». Запрос генерируется с помощью команды *Select*. Различие между обычной транзакцией и запросом заключается в том, что при запросе данные не изменяются.

Индекс — структура, которая позволяет извлекать данные быстро и эффективно (точно также, как содержание какой-либо книги позволяет найти интересующий раздел). Индекс объявляется для одного или нескольких столбцов. Доступ к таблице происходит по проиндексированному столбцу (столбцам).

Кластер — набор таблиц, которые физически хранятся как одна и имеют общие столбцы. Использование кластеров крайне эффективно, если часто обрабатываются запросы к данным двух и более таблиц, имеющих общие столбцы. К таблицам можно обращаться по отдельности даже в том случае, если они являются частью кластерной таблицы.

Конкурирование (*concurrency*) — это способность программы выполнять несколько функций одновременно. Применительно к Oracle *конкурирование* — это возможность одновременного доступа к данным для множества пользователей.

Контрольная точка (*Checkpoint*) — операция, приводящая к тому, что все измененные данные (блоки данных в памяти) записываются на диск. Это ключевой фактор в проблеме быстрого восстановления базы данных после сбоя.

Кэш буферов данных — область памяти для быстрого доступа к данным. С точки зрения аппаратного обеспечения — это небольшой (применительно к оперативной памяти) объем памяти, который значительно быстрее основной памяти. Этот объем памяти используется для снижения времени, необходимого на частую загрузку данных или инструкций в центральный процессор (ЦП). ЦП сам по себе содержит встроенный кэш.

Объекты схемы — это абстракция (логическая структура) составляющих базы данных. Объекты схемы состоят из индексов, кластеров, пакетов, последовательностей, хранимых процедур, синонимов, таблиц, представлений и т.д.

Последовательность, генератор последовательностей — используется для создания последовательности цифр, хранимых в кэш буферов данных.

Представление (*вид*) — это рамка, окно для просмотра данных из одной или более таблиц. Вид не хранит никаких данных, он только представляет их. С видами можно делать те же операции, что и с таблицами (строить запросы, обновлять, удалять), без всяких ограничений. Представления часто используются, чтобы упростить восприятие пользователем хранящихся в базе данных путем извлечения из таблицы лишь части необходимых данных или набора данных из нескольких таблиц. Кроме того, представления могут использоваться для ограничения доступа пользователей к некоторым данным.

Программный блок — относительно СУРБД Oracle это программа, используемая для описания пакета, хранимой процедуры или последовательности процедур.

Процедура — это набор SQL или PL/SQL-команд, который выполняет определенную задачу. Процедура может иметь входные параметры, но не имеет выходных.

Словарь данных (*Data Dictionary*) — набор таблиц, используемых для поддержания информации о БД.

Схема (*Schema*) — коллекция объектов БД.

Таблица — основная единица хранения данных БД Oracle. Таблица состоит из имени, строк и столбцов. Каждый столбец также имеет имя и тип данных. Таблицы хранятся в табличных пространствах, причем часто в одном табличном пространстве находятся несколько таблиц.

Транзакция — логически завершенный фрагмент последовательности действий (одна или более SQL-команд, завершенных фиксацией или откатом).

Триггер — это механизм, позволяющий создавать процедуры, которые будут автоматически запускаться при выполнении команд *Insert*, *Update* или *Delete*.

Узкое место (*Bottleneck*) — компоненты, ограничивающие производительность или эффективность системы.

Функция — это также совокупность SQL или PL/SQL-команд, которая реализует определенную задачу. Функция отличается от процедуры тем, что возвращает какое-либо значение переменной (процедура ничего не возвращает). Создание функций позволяет уменьшить число инструкций, передаваемых по сети.

Хранимая процедура — это SQL-запрос, хранимый в словаре данных. Хранимые процедуры разрабатываются для эффективного выполнения запросов. При использовании хранимых процедур можно уменьшить сетевой трафик СУРБД и тем самым увеличить производительность.

Чистый буфер (*clean buffer*) — это такой буфер, содержимое которого не было подвержено изменению. Так как этот буфер не изменился, то процессу DBWR нет необходимости записывать его на жесткий диск.

DBWR (*DataBase WRiter*) — процесс, основная задача которого — записывать изменения базы данных на физический жесткий диск.

DDL (*Data Definition Language*) — язык описания данных. Команды этого языка предназначены для создания, изменения и удаления объектов базы данных. В системе Oracle команды DDL связаны с администрированием баз данных. Перед и после выполнения каждой DDL-команды система Oracle обязательно фиксирует все текущие транзакции (чтобы избежать потери информации).

DML (*Data Manipulation Language*) — язык манипулирования данными. Команды этого языка позволяют строить запросы и оперировать с данными существующих объектов схемы. В отличие от DDL фиксирование транзакций после каждой команды не производится. Существуют следующие команды DML: *Delete*, *Insert*, *Select* и *Update*; *Explain plan*-команды; и *Lock table*-команды.

SGA (*System Global Area*) — разделяемая область памяти, используемая для хранения данных и управляющей информации экземпляра Oracle. SGA размещается в памяти при запуске экземпляра Oracle и освобождается при завершении работы. SGA составляют буферы данных, буфер журнала изменений и разделяемый пул (shared pool).

12.3. Конфигурации Oracle

Существует много видов конфигураций. Рассмотрим некоторые из них.

Веб-сервер — предназначен для работы со статическими и динамическими веб-страницами. Эти страницы могут быть как очень простыми, так и комплексными, генерируемыми из баз данных. Веб-сервер Oracle, как правило, используется для коммерческих веб-приложений. Такие приложения позволяют покупателям просматривать каталоги, которые содержат изображения товаров и даже видеоиллюстрации. Покупатель может приобрести понравившийся ему товар. Характерные черты веб-сервера Oracle заключаются в том, что он обычно поддерживает значительное количество пользователей и содержит большие объемы данных. Производительность веб-сервера зависит от объема оперативной памяти.

Видео-сервер — предназначен для обработки видеоинформации. Характерной особенностью видеосервера является то, что он должен иметь широкую полосу пропускания, чтобы поддерживать большое количество видеопотоков. Видеосервер должен справляться с большой нагрузкой ввода (вывода), так как при чтении с устройств загружаются сразу большие блоки данных.

Информационная лавка (*Data Mart*) — это уменьшенная версия хранилища данных (Data Warehouse), как правило, ориентирован-

ная на решение узкоспециализированных задач. Она обеспечивает хранение и обработку информации, требующей менее сотни гигабайт памяти.

Характерные черты DSS — это выполнение множества запросов, связанных с обработкой больших объемов информации, хранящихся в разных таблицах и разных базах данных.

Хранилище данных (*Data Warehouse*) — это крупномасштабная система, которая хранит результаты работы OLTP и DSS-систем. Эта система должна обеспечить хранение и обработку информации, занимающей многие сотни гигабайт памяти.

DSS (*Decision Support System*) — это системы поддержки принятия решений, которые используются в процессах интеллектуального анализа данных.

OLAP (*On-line Analytical Processing*) — система аналитической обработки информации в реальном масштабе времени. Как правило, пользователи систем OLAP — это финансовые аналитики или маркетинговый персонал, работающий с данными на глобальном уровне.

OLTP (*On-line Transaction Processing*) — это система оперативной обработки транзакций.

Характерные черты OLTP-систем заключаются в том, что они обеспечивают работу большого числа пользователей, работающих с многопользовательскими базами данных. Так как пользователи ждут возвращения данных на свои запросы, то система должна как можно быстрее обеспечивать ответы на запросы всех клиентов. OLTP-системы для этого и предназначены. Работа OLTP-систем должна обеспечивать с высокую скорость соответствующих процессов обработки информации.

12.4. Типы пользователей

Типы пользователей и их обязанности могут отличаться в зависимости от конфигурации Oracle и конкретной организации корпоративной базы данных. В крупных системах, например, обязанности администратора базы данных могут распределяться среди нескольких специалистов, а в мелких системах один человек может выполнять функции нескольких типов пользователей одновременно.

Можно выделить следующие основные типы пользователей, характерные для всех систем управления базами данных:

- администраторы баз данных;
- администраторы по защите данных;
- разработчики приложений;
- администраторы приложений;
- пользователи базы данных;
- администраторы сети.

12.5. Администратор базы данных

DataBase Administrator (*DBA*) — это специалист, управляющий работой базы данных. Обычно обязанности DBA подразделяют на две категории: основные и дополнительные.

Основные обязанности DBA состоят из следующих задач.

- Установка нового программного обеспечения. Эта обязанность DBA заключается в установке новых версий Oracle, приложений и другого программного обеспечения, относящегося к администрированию СУБД. Эта задача предусматривает также обязательное тестирование устанавливаемых программ перед введением их в рабочую среду.
- Конфигурация программного и аппаратного обеспечения. В большинстве случаев доступ к настройке программного и аппаратного обеспечения имеет только системный администратор, поэтому DBA должен производить установку программ, конфигурирование программного и аппаратного обеспечения только совместно с системным администратором.
- Обеспечение безопасности. Это одна из основных обязанностей DBA. Управление безопасностью и администрирование включают в себя добавление и удаление пользователей, управление квотами, аудит и разрешение проблем безопасности.
- Настройка производительности. DBA должен постоянно проверять производительность системы, а при необходимости выполнять ее перенастройку. Даже хорошо настроенная система нуждается в постоянной проверке и периодической перенастройке. Иногда достаточно изменить параметры системы, иногда — изменить индексы, а может, и перестроить структуру таблиц.
- Резервное копирование и восстановление системы. Возможно это одна из главных задач DBA — постоянно сохранять данные в системе. Чтобы делать это эффективно, необходимо разработать процедуру резервного копирования и стратегию восстановления данных. Очень важно периодически тестировать отработанную схему резервного копирования и восстановления данных.
- Процедура постоянного (планового) обслуживания. Обслуживание СУБД лучше всего производить в ранние часы либо по выходным дням, чтобы не нарушать работу пользователей. В обслуживание входят: архивирование, тестирование и настройка системы. Для осуществления планового обслуживания системы администратор должен составить календарь обслуживания СУБД и довести его для сведения клиентам.
- Локализация неисправностей и восстановление системы после сбоя. В обязанности DBA входит восстановление работоспособности СУБД в случае ее сбоя. Поскольку сбой системы приводит к тому, что пользователи могут потерять доступ к своим данным, DBA обязан как можно быстрее восстановить работу системы. Хо-

рошо подготовленный DBA должен предусмотреть и заранее составить планы восстановления системы после сбоев.

Дополнительные обязанности DBA сводятся, как правило, к оказанию помощи отдельным клиентам и могут включать в себя следующие задачи администрирования.

- Анализ данных. DBA должен провести анализ данных и дать отдельным разработчикам или пользователям рекомендации по улучшению производительности или эффективности хранения данных.
- Разработка БД (предварительная). DBA может участвовать на предварительной стадии разработки структуры БД. Поскольку DBA знает систему изнутри, он может указать команде разработчиков на потенциальные проблемы и помочь в увеличении производительности программ.
- Оказание консультаций разработчикам по хранимым SQL-процедурам. DBA должен быть готов стать консультантом для разработчиков и пользователей. DBA довольно часто привлекается к разрешению проблем SQL-кода и разработке (написанию) хранимых процедур.
- Разработка производственных стандартов и соглашений по именам. Хотя данная задача и отнесена к разряду дополнительных, она играет одну из основных организационных проблем управления. Учитывая что в разработке и развертывании приложений могут принимать участие несколько различных групп, DBA должен принимать активное участие, а то и играть главную роль в разработке производственных стандартов и соглашений по именам, чтобы приложения соответствовали этим стандартам.
- Документирование среды. DBA должен документировать каждый аспект среды СУБД, включая конфигурацию оборудования, обновления и изменения программного обеспечения и СУБД, а также все вопросы, связанные с изменением системы и ее параметров. DBA должен уметь полностью восстановить систему по документации в случае необходимости.
- Планирование нагрузки системы и необходимого объема памяти. Неотъемлемой частью работы DBA является определение необходимости в приобретении дополнительных серверов, дополнительной дисковой и оперативной памяти, чтобы удовлетворить возросшие потребности пользователей. Прогнозируя ожидаемую потребность в аппаратных средствах, администратор обеспечивает надежность работы информационной системы предприятия.

12.6. Физическая архитектура хранения данных

СУРБД Oracle предназначена для одновременного доступа к большим объемам хранимой информации (терабайты). СУРБД складывается из двух составляющих: база данных (информация) и экземпляр (конкретная реализация системы).

База данных состоит из физических файлов, хранящихся в системе, и логических частей (например, схема БД). Физические файлы — это файлы, которые хранятся на диске, а логические файлы являются компонентами физического уровня.

Итак, базы данных Oracle состоят из двух уровней: физического и логического.

Физический уровень. Он включает в себя три категории файлов: файлы данных, файлы журналов операций (redo log files), управляющие файлы.

Файлы данных. В этих файлах хранится информация, имеющаяся в БД. Это может быть как один файл данных, так и сотни таких файлов. Информация из одной таблицы может быть распределена по нескольким файлам данных (а несколько таблиц могут делить между собой пространство файлов данных). Распределение таблиц по нескольким файлам данных может значительно увеличить производительность системы. Число файлов данных ограничено параметром *Maxdatafiles*.

Файлы журналов операций. Это файлы, которые содержат информацию, необходимую для процесса восстановления в случае сбоя системы, и хранят все изменения, произошедшие в базе данных. С помощью журнала операций восстанавливаются те изменения, которые были произведены, но не зафиксированы перед сбоем системы. Файлы журналов операций должны быть очень хорошо защищены от аппаратных сбоев (как на программном, так и на аппаратном уровне). Если информация журнала операций будет утеряна, то восстановить систему будет практически невозможно.

Управляющие файлы. Управляющие файлы содержат информацию, необходимую для запуска экземпляра Oracle, в том числе расположение файлов данных и файлов журналов операций. Управляющие файлы должны быть хорошо защищены.

Логический уровень. Логический уровень составляют следующие элементы:

- табличные пространства;
- схема БД.

Табличные пространства. База данных может состоять из одной и более логических частей, называемых табличными пространствами. Табличные пространства используются для логической группировки данных между собой. Например, можно определить одно табличное пространство для бухгалтерских данных, а другое — для складских. Сегментирование групп по табличным пространствам упрощает администрирование этих групп. Каждое табличное пространство состоит из одного или более файлов данных. Используя несколько файлов данных для одного табличного

пространства, можно распределить их по разным дискам, увеличив тем самым скорость ввода (вывода) и соответственно производительность системы.

Схема БД. В СУРБД Oracle контроль над дисковым пространством осуществляется с использованием специальных логических структур — схем баз данных. Эти структуры состоят из блоков данных, экстентов, сегментов.

Блок данных — это наименьшая единица хранения данных в БД Oracle. Блок данных содержит заголовочную информацию о себе и данные. Они физически хранятся на диске. Блоки данных в большинстве систем занимают 2Кб (2048 байт), для увеличения эффективности работы системы это значение можно изменить.

Экстент состоит из блоков данных. Экстенты являются строительными блоками сегментов и в то же время состоят из блоков данных. Экстенты используются для минимизации неиспользуемого (пустого) пространства хранилища. По мере увеличения числа данных в табличных пространствах экстенты используются для хранения тех данных, которые могут разрастаться. Таким образом, несколько табличных пространств могут делить между собой пространство хранилища без предопределения разделов этих табличных пространств.

При создании табличного пространства можно указать минимальное число определения экстент, а также число экстент, добавляемых при заполнении уже определенных экстент. Такое распределение позволяет контролировать все пространство хранилища БД.

Сегмент, в свою очередь, состоит из совокупности экстентов, содержащих определенный вид данных. БД Oracle использует четыре типа сегментов:

- сегмент данных — хранит пользовательские данные;
- индексный сегмент — содержит индексы;
- сегмент отката — хранит информацию отката, используемую при возврате к предыдущему состоянию БД;
- временный (промежуточный) сегмент — создается, если для выполнения SQL-выражения необходимо дополнительное рабочее пространство. Эти сегменты уничтожаются сразу после выполнения SQL-команд. Промежуточные сегменты используются также в разнообразных операциях с БД, например при сортировке.

Экземпляр

Экземпляр представляет собой конкретный способ доступа к данным, который состоит из разделяемой памяти и процессов.

Разделяемая память. Oracle использует разделяемую память (shared memory) в различных целях, таких как кэширование данных и индексов, хранение программного кода.

Разделяемая память подразделяется на несколько частей (или структур памяти). Основными структурами памяти Oracle являются:

- системная глобальная область;
- программная глобальная область.

Системная глобальная область (SGA — System Global Area). Это область разделяемой памяти, которую Oracle использует для хранения данных и управляющей информации одного конкретного экземпляра Oracle. SGA размещается в памяти при запуске экземпляра Oracle и освобождает память при останове. Каждый запущенный экземпляр Oracle имеет свою собственную SGA.

Информация в SGA состоит из следующих компонентов (каждый из которых создается в памяти при запуске экземпляра): кэш буферов БД, буфер журнала изменений, разделяемый пул.

Кэш буферов БД хранит последние открытые блоки данных. Эти блоки могут содержать данные, которые изменились, но еще не были записаны на диск (грязные блоки); данные, которые не изменялись либо были записаны на диск после изменения (чистые блоки). Так как кэш буферов БД хранит блоки данных на основе алгоритма последних используемых блоков, то наиболее активно используемые блоки постоянно остаются в памяти (тем самым снижая дисковый ввод (вывод) и увеличивая производительность системы).

Буфер журнала изменений хранит данные об изменениях БД. Буфер журнала изменений записывается в файл журнала изменений настолько быстро и эффективно, насколько это возможно. (Журнал изменений используется для восстановления экземпляра СУБД Oracle в случае сбоя системы.)

Разделяемый пул — это область SGA, в которой хранятся такие структуры разделяемой памяти, как разделяемые SQL-области в библиотечном кэше и внутренняя информация словаря данных. Разделяемый пул состоит из библиотечного кэша и кэша словаря данных.

Библиотечный кэш используется для хранения разделяемых SQL-выражений. Здесь для каждого уникального SQL-выражения строится дерево разбора строк и план исполнения, которые кэшируются (т. е. сохраняются в библиотечном кэше). Если несколько приложений отправляют одинаковые SQL-выражения, то для ускорения работы используется разделяемая SQL-область (так как используются уже разобранные строки и готовый план исполнения, то происходит экономия времени).

Кэш словаря данных содержит набор таблиц и представлений, используемых в качестве ссылок к БД Oracle. Здесь хранится информация о логической и физической структуре БД. Словарь данных содержит следующую информацию:

- пользовательская информация (например, пользовательские привилегии);

• ограничения целостности, определенные для таблиц БД;
• имена и типы данных всех столбцов таблиц БД;
• информация об объеме памяти, определенном и используемом объектами схемы данных.

Для обеспечения высокой производительности необходимо установить достаточный объем памяти под кэш словаря данных.

Программная глобальная область (PGA — Program Global Area). Это такая область памяти, в которой хранятся данные и управляющая информация о серверных процессах Oracle. Размер и содержание PGA определяются опциями, которые задаются при инсталляции Oracle. Эта область состоит из следующих компонентов:

• пространство стека — это память, хранящая переменные сеансов, массивы сеансов и т.д.;
• информация сеанса — если Oracle работает не в мультинитиевом режиме, то информация сеанса хранится в PGA. В противном случае информация сеанса хранится в SGA;
• приватная SQL-область — это часть PGA, в которой хранятся связанные переменные, и буферы реального времени.

Процесс. Процесс (или нить) — это механизм выполнения программного кода, который может выполняться незаметно для пользователя. Кроме того, несколько процессов могут работать одновременно. В разных операционных системах и на разных платформах они могут называться по-разному (процессы, нити, домены и т.д.), но, в сущности, одинаковы. СУРБД Oracle работает с двумя видами процессов: пользовательские процессы и процессы Oracle, также известные как фоновые, или теневые. В некоторых операционных системах (таких как Windows NT) процессы действительно являются нитями, но чтобы не путаться в понятиях, будем называть их просто процессами.

Пользовательские процессы. Пользовательские (клиентские) процессы — это пользовательские соединения с СУРБД. Пользовательский процесс управляет вводом и взаимодействует с серверными процессами Oracle через программный интерфейс Oracle. Пользовательский процесс используется также для выдачи информации пользователю и при необходимости представляет ее в более удобной форме.

Процессы Oracle. Процессы Oracle выполняют функции для пользовательских процессов. Эти процессы могут быть разбиты на две группы: серверные процессы (выполняющие функции для активных процессов) и фоновые процессы (выполняющие функции СУРБД в целом).

Серверные процессы (теневые) взаимодействуют между процессами пользовательскими и Oracle, исполняя пользовательские запросы. Например, если пользовательский процесс запрашивает часть данных, которых еще нет в SGA, то теневой процесс несет ответ-

ственность за чтение блоков данных из БД в SGA. Между пользовательским и теневым процессом возникает связь «один к одному», хотя один теневой процесс может одновременно взаимодействовать с несколькими пользовательскими (конфигурация мультинитевого сервера), экономя системные ресурсы.

Фоновые процессы используются для выполнения разнообразных задач СУРБД Oracle. Эти задачи варьируются от взаимодействия с экземпляром Oracle до записи грязных блоков на диск. Существует девять фоновых процессов Oracle:

- DBWR (DataBase WRiter) ответственен за запись грязных блоков из блоковых буферов БД на диск. Когда транзакция изменяет информацию в блоке данных, этот блок данных не обязан быть немедленно записан на диск. Следовательно, DBWR может записывать данные на диск способом более эффективным, чем запись всех изменений по отдельности. DBWR обычно записывает данные тогда, когда они уже нужны для чтения. Записываются также те данные, которые были недавно использованы. Для систем с асинхронным вводом (выводом) достаточно одного процесса DBWR. Для остальных систем можно значительно увеличить производительность, создав несколько процессов DBWR;
- LGWR (LoG WRiter) записывает данные из журнального буфера в журнал изменений;
- CKPT (CheckK PoinT) дает сигнал процессам DBWR о необходимости выполнения контрольной точки и обновления всех файлов данных и управляющих файлов. Контрольная точка — это событие, когда все измененные буферы БД записываются на диск. CKPT — это не обязательный процесс. Если процесс CKPT не запущен, то его работу принимает на себя LGWR;
- PMON (Process MONitor) используется для поддержания остальных процессов и перезапуска преждевременно погибших. Также PMON очищает неиспользуемые области буферов и освобождает те ресурсы, которые могут быть еще заняты. Ответственен за перезапуск всех зависших процессов и диспетчеров;

SMON (System MONitor) выполняет восстановление экземпляра при его запуске. Это включает очистку временных сегментов и восстановление незаконченных транзакций, а также дефрагментирует БД;

RECO (RECOvery) очищает незаконченные транзакции в распределенной БД. Выполняет фиксацию или откат спорных транзакций;

ARCH (ARCHiver) копирует файлы журнала изменений при их заполнении. ARCH активен только в том случае, если СУРБД работает в режиме *Archivelog*. Если система не работает в этом режиме, то возможны ситуации, в которых не удастся восстановить систему после сбоя. В некоторых случаях все же можно работать и в режиме *Noarchivelog*;

LCKn (Parallel Server LoCK) — до 10 процессов (где *n* — значения от 0 до 9) могут использоваться при работе сервера в параллельном режиме. Выполняют функции межэкземплярной блокировки;

Dnnn (Dispatcher) — при работе сервера в мультинитевом режиме существует хотя бы один диспетчерский процесс, ответственный за каждый протокол взаимосвязи. Диспетчерские процессы организуют взаимодействие между пользовательскими и разделяемыми серверными процессами.

12.7. Транзакции. Принципы работы

Транзакция — это одна или более SQL-команд, завершенных фиксацией (commiting) или откатом (rollbacking). Под *фиксацией* понимается принятие и сохранение всех изменений. *Откат* — это процедура отмены последних изменений, т.е. возврат к предыдущему состоянию БД.

Чтобы понять как работает система Oracle, рассмотрим по шагам пример работы простой транзакции.

1. Приложение обрабатывает пользовательский ввод и создает соединение с сервером посредством SQL*Net.

2. Сервер принимает запрос на соединение и создает серверный процесс.

3. Пользователь выполняет SQL-команду (или совокупность команд). В нашем примере будем считать, что пользователь изменяет данные в строке таблицы.

4. Серверный процесс просматривает разделяемый пул — есть ли там SQL-область с идентичными SQL-командами. Если он находит аналогичную разделяемую SQL-область, то серверный процесс проверяет права пользователя на доступ к данным. Предположим, что права есть, тогда серверный процесс выполняет команды, используя разделяемую SQL-область. Однако если разделяемая SQL-область не найдена, то выделяется память под новую, а затем происходит разбор и выполнение SQL-команд.

5. Серверный процесс ищет данные в SGA (если они там есть) или считывает их из файла данных в SGA.

6. Серверный процесс изменяет данные в SGA. Серверный процесс может только читать данные из файла данных. Позже процесс DBWR запишет измененные блоки данных в постоянное хранилище (жесткий диск, магнитная лента).

7. Пользователь выполняет команду *Commit* (фиксация) или *Rollback* (откат). *Commit* завершает транзакцию, а *Rollback* отменяет изменения. Если транзакция зафиксирована, то процесс LGWR немедленно записывает ее в файл журнала изменений.

8. Если транзакция успешно завершена, то клиентскому процессу передается код завершения. Если произошел какой-либо сбой, то возвращается сообщение об ошибке.

Примечание. Транзакция не считается зафиксированной до тех пор, пока не завершена запись в файл журнала изменений (redo log file). Этот механизм способствует тому, что в случае сбоя зафиксированная транзакция может быть восстановлена.

12.8. Обзор функций Oracle

При работе с СУРБД Oracle необходимо обеспечить выполнение таких задач, как обеспечение целостности данных, восстановление базы данных после сбоев, перехват ошибок и т.д. Это осуществляется следующими функциями: создание контрольных точек (checkpointing), журналирование и архивирование.

Создание контрольных точек. Сигнал к созданию контрольной точки поступает либо от процесса DBWR, либо от LGWR. Но что же такое контрольная точка и для чего она необходима?

Так как все изменения блоков данных происходят в блоковых буферах, то изменения данных в памяти не обязательно отражаются в этих блоках на диске.

Процесс кэширования происходит по алгоритму последнего использованного блока, поэтому буфер, подверженный постоянным изменениям, помечается как последний использованный и процесс DBWR не записывает его на диск. Контрольная точка применяется для того, чтобы эти буферы были записаны на диск. Все грязные буферы вынуждены быть сохранены на диске в обязательном порядке.

Контрольная точка может выполняться в двух режимах: нормальная контрольная точка и быстрая контрольная точка.

В режиме *нормальной контрольной точки* грязные буферы записываются последовательно процессом DBWR. В этом режиме контрольная точка выполняется гораздо дольше, но затрагивает меньше системных ресурсов, чем быстрая.

В режиме *быстрой контрольной точки* DBWR записывает одновременно некоторое число буферов. Такая контрольная точка выполняется очень быстро. Она более эффективна в смысле работы ввода (вывода), но в тоже время значительно снижает производительность системы.

Частое выполнение контрольных точек способствует увеличению времени, необходимого для восстановления системы в случае сбоя. Контрольная точка автоматически выполняется при смене журнала изменений.

Журналирование и архивирование. Журнал изменений (redo log) записывает все изменения БД Oracle. Целью создания журнала изменений является возможность экстренного восстановления БД в случаях сбоев системы и потери файлов данных. Восстановив файлы данных из ранее сделанных резервных копий, файлы журнала изменений (включая архивные файлы журнала) могут повторить

все последние транзакции. Таким образом, файлы данных будут полностью восстановлены.

Когда файл журнала изменений оказывается полностью заполненным, происходит смена журнала и процесс LGWR заводит новый файл. Во время смены журнала процесс ARCH записывает заполненный файл в архив файлов журналирования. В тот момент когда архивирование только закончилось, файл журнала изменений помечается как доступный. Очень важно, чтобы архивные файлы журнала изменений надежно хранились, так как они могут понадобиться для восстановления системы.

12.9. Триггеры в Oracle

Для создания триггеров, хранимых процедур и просто скриптов (в Oracle их принято называть безымянными блоками) в Oracle создан свой язык. Этот язык получил название PL/SQL (Program Language SQL).

Для каждой таблицы можно создать до 12 триггеров. Вот шаблон триггера:

```
CREATE TRIGGER [name] (событие вызова триггера)...
(необязательное ограничение триггера)
BEGIN
(действие триггера)
END;
```

При определении триггера можно указать, сколько раз он должен выполняться: для каждой изменяемой строки (row trigger) либо однократно для всего выполняемого выражения, независимо от того, сколько строк будет изменено (statement trigger).

Row trigger — часто используемый вид триггера. Выполняются для каждой строки по одному разу. Например, если SQL-выражение UPDATE обновляет множество строк в таблице, то триггер вызывается для каждой строки, которая изменяется выражением UPDATE. Если выражение не влияет ни на одну строку, то триггер вызываться не будет вообще.

Statement trigger — вызывается независимо от числа измененных строк в таблице, даже если не изменялась ни одна строка. Например, если выражение DELETE удаляет из таблицы несколько строк, то триггер уровня выражения DELETE вызывается только единожды, независимо от того, сколько строк удаляется из таблицы.

При определении триггера необходимо указать момент выполнения (trigger timing) тела триггера — до или после выражения. BEFORE и AFTER применимо как к триггерам выражений, так и для строчных триггеров.

INSTEAD-OF (вместо) — еще один тип триггера, который поддерживает Oracle. Триггеры INSTEAD-OF доступны только в ре-

дакции Oracle8i. Они могут использоваться в многотабличных и объектных представлениях. В отличие от других триггеров они применяются вместо выполнения DML-выражений (INSERT, UPDATE и DELETE).

Представление можно модифицировать как обычную таблицу с помощью INSERT, UPDATE и DELETE, и для соответствующего изменения будет запущен триггер INSTEAD-OF. Триггеры INSTEAD-OF активизируются для каждой изменяемой строки.

Для демонстрации возможностей триггеров INSTEAD-OF создадим триггер (view manager_info), который перечисляет всех менеджеров каждого отдела:

```
CREATE VIEW manager_info AS
SELECT d.deptno, d.deptname, e.empno, e.empname
FROM emp e, dept d
WHERE e.empno = d.manager_num;
```

Теперь определим триггер INSTEAD-OF, который будет обрабатывать вставку в представлении. Вставка в view manager_info может быть транслирована в операцию обновления колонки *manager_num* таблицы *dept*.

В триггере можно определить ограничение, указывающее, что в отделе, представленном менеджером отдела, должен в данный момент работать по крайней мере один работник.

```
CREATE TRIGGER manager_info_insert
INSTEAD OF INSERT ON manager_info
--информация о новом менеджере
REFERENCING NEW AS n
FOR EACH ROW
DECLARE
empCount NUMBER;
BEGIN
/* Первая проверка, выполняется для подтверждения того, что
число работников отдела больше одного */
SELECT COUNT(*) INTO empCount
FROM emp e
WHERE e.deptno =:n.deptno;
/* Если работников достаточно, то сделать его (ее) менеджером */
IF empCount >= 1 THEN
UPDATE dept d
SET manager_num = :n.empno
WHERE d.deptno = :n.deptno;
END IF;
END;
```

Контрольные вопросы

1. Что такое триггер?
2. Что такое транзакция?
3. Какие категории файлов включает в себя физический уровень БД Oracle?
4. В каком табличном пространстве хранится словарь данных?
5. Какая разница между сегментом, экстентом и блоком данных?
6. Какие основные элементы составляют экземпляр Oracle?

ЧАСТЬ IV

ПОСТРЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ

Глава 13

ОРИЕНТАЦИЯ НА РАСШИРЕННУЮ РЕЛЯЦИОННУЮ МОДЕЛЬ

13.1. Основные направления совершенствования реляционных баз данных

Рассмотрим основные направления в области исследований и разработок систем управления — так называемых постреляционных баз данных.

Можно отметить три направления в области развития СУБД следующего поколения.

Первое направление связано с максимальным использованием существующих технологий управления и организации реляционных СУБД с дальнейшим совершенствованием систем управления внешней памятью.

Второе направление связано с созданием генераторов системы управления в виде наборов модулей со стандартизованными интерфейсами.

Третье направление развития СУБД по сути является синтезом первых двух направлений. СУБД проектируется как некоторый интерпретатор системы правил и набор модулей-действий, вызываемых в соответствии с этими правилами. Для таких систем разрабатываются специальные языки формирования правил.

Можно сказать, что СУБД следующего поколения — это прямые наследники реляционных систем. Тем не менее различные направления систем третьего поколения стоит рассмотреть отдельно, поскольку они обладают разными характеристиками.

Одним из основных положений реляционной модели данных является требование нормализации отношений. Это позволяет проектировать БД с предельно понятной структурой и экономить значительные ресурсы памяти компьютера. В этом случае для обеспечения целостности информации используются соответствующие связи между таблицами.

Однако реляционные СУБД стали применять не только в сфере бизнеса для управленческих задач, но и в сфере промышленного производства (CALS-технологии). Применение реляционных баз данных оказалось весьма эффективным для разработки систем

автоматизированного проектирования технологических процессов (САПРТП), разработке экспертных систем и других задачах управления и технической подготовки производства.

Такие системы обычно оперируют сложноструктурированными объектами — некоторым комплексом логически связанных таблиц, для анализа информации которых, а тем более для выбора рациональных технических решений приходится выполнять сложные запросы.

В соответствии с такими задачами применения реляционных БД появилось новое направление их развития. Суть этого направления сводится к тому, что в системах управления базами данных формируются сложные объекты, объединяющие в себе не только исходные таблицы БД, но и соответствующие запросы.

При этом сохраняется четкая граница между логическим и физическим представлениями таких объектов. В частности, для любого сложного объекта (произвольной сложности) обеспечивается возможность перемещения или копирования его как единого целого из одной части базы данных в другую ее часть или даже в другую базу данных.

Это очень обширная область исследований, в которой затрагиваются вопросы разработки моделей данных, структур данных, языков запросов, управления транзакциями, журнализации и т.д.

Это новое направление в разработке СУБД хотя и основывается на реляционной модели, но в ней не обязательно поддерживается требование полной нормализации отношений.

С расширением области применения реляционного подхода для решения задач, особенно в направлении CALS-технологий, стало очевидным, что применение принципа нормализации таблиц БД и создание отдельных транзакций и процедур при выполнении различных запросов «сводит на нет» все преимущества нормализованной схемы организации базы данных.

В ненормализованных реляционных моделях данных допускается создание и хранение в качестве объекта (исходного элемента системы) кортежей (записей), массивов (регулярных индексированных множеств данных), регулярных множеств элементарных данных, а также отношений. При этом такая вложенность может быть неограниченной.

Системы управления базами данных, в которых формируются такие сложные объекты, называют *объектно-ориентированными базами данных* (*ООБД*).

Очевидно, что такие системы должны обязательно иметь в своем составе языки программирования для создания и управления такими объектами. Кроме того, они должны обрабатывать различные типы данных, в том числе формируемые пользователями.

В 1995 г. компания Sun Microsystems объявила о выпуске нового продукта — языка из семейства интерпретаторов под названием

Java. Язык Java является расширенным подмножеством языка Си++. Основные изменения касаются того, что язык является пооператорно интерпретируемым (в стиле языка Basic), а программы, написанные на языке Java, гарантированно безопасны (в частности, при выполнении любой программы не может быть поврежден интерпретатор). Для этого, в частности, из процедур языка удалены математические действия над указателями. В то же время Java остается мощным объектно-ориентированным языком, включающим в себя развитые средства определения абстрактных типов данных. Компания Sun Microsystems продвигает язык Java с целью расширения возможностей службы «всемирной паутины» (World Wide Web) Интернет. Основная идея состоит в том, что с сервера WWW клиентам передаются не данные и не результаты обработки данных, а объекты, методы выполнения которых запрограммированы на языке Java и выполняются на стороне клиента.

13.2. Генерация систем баз данных, ориентированных на приложения

Появление данного направления в развитии СУБД определяется тем, что невозможно создать универсальную систему управления базами данных, которая будет достаточна и не избыточна для применения в любом приложении. Например, если посмотреть на использование существующих СУБД для решения практических задач в производстве и бизнесе, то можно утверждать, что в большинстве случаев применяется не более чем 30 % возможностей системы. Тем не менее приложение несет всю тяжесть поддерживающей его СУБД, рассчитанной на использование в наиболее общих случаях.

Поэтому очень заманчиво производить не законченные универсальные СУБД, а нечто вроде компиляторов (compiler compiler), позволяющих собрать систему баз данных, ориентированную на конкретное приложение (или класс приложений). Рассмотрим простые примеры.

Так, в системах резервирования проездных билетов запросы обычно настолько просты (например: «выдать очередное место на рейс № 645»), что нет смысла производить широкомасштабную оптимизацию запросов. Однако информация, хранящаяся в базе данных, настолько критична (кто из нас не сталкивался с проблемой наличия двух или более билетов на одно место?), что особенно важны гарантированные синхронизация обновлений базы данных и ее восстановление после любого сбоя.

В статистических системах запросы могут быть произвольно сложными (например: «выдать число холостых мужчин, проживающих в России и имеющих не менее трех зарегистрированных детей»), что вызывает необходимость использования развитых

средств оптимизации запросов. Но поскольку речь идет о статистике, здесь не требуется поддержка строгой сериализации транзакций и точного восстановления базы данных после сбоев. (Поскольку речь идет о статистической информации, потеря нескольких ее единиц обычно не существенна.)

Поэтому желательно уметь генерировать систему баз данных, возможности которой в достаточной степени соответствуют потребностям приложения. На сегодняшний день на коммерческом рынке такие генерационные системы отсутствуют (например, при выборе сервера системы Oracle при разработке конкретного приложения нельзя отказаться от каких-либо свойств системы).

13.3. Оптимизация запросов, управляемых правилами

Под оптимизацией запросов в реляционных СУБД обычно подразумевают такой способ обработки запросов, при котором по начальному представлению запроса путем его преобразований вырабатывается оптимальный процедурный план его выполнения.

Соответствующие преобразования начального представления запроса выполняются специальным компонентом СУБД — оптимизатором, и оптимальность производимого им плана выполнения запроса носит субъективный характер, поскольку критерий оптимальности заложен разработчиком в оптимизатор.

Основная неприятность, связанная с оптимизаторами запросов, заключается в том, что отсутствует принятая технология их программирования. Обычно оптимизатор представляет собой некоторый набор относительно независимых процедур, которые жестко связаны с другими компонентами компилятора. По этой причине очень трудно менять стратегии оптимизации или качественно их расширять (делать это приходится, поскольку оптимизация вообще и оптимизация запросов являются эмпирической дисциплиной, а хорошие эмпирические алгоритмы появляются только со временем).

Решают эту проблему при помощи компромиссных решений, не выходящих за пределы традиционной технологии производства компиляторов. В основном все они связаны с применением тех или иных инструментальных средств, обеспечивающих автоматизацию построения компиляторов. Такие инструментальные средства имеются, например, в системах DB2, Oracle, Informix.

13.4. Поддержка динамической информации и темпоральных запросов

Обычные реляционные БД хранят мгновенный снимок модели предметной области. Любое изменение в момент времени *t* некоторого объекта приводит к недоступности состояния этого объек-

та в предыдущий момент времени. На самом деле в большинстве развитых СУБД предыдущее состояние объекта сохраняется в журнале изменений, но возможности доступа к ним со стороны пользователя нет.

Конечно, можно явно ввести в хранимые отношения явный временной атрибут и поддерживать его значения на уровне приложений. В большинстве случаев так и поступают. Так, в стандарте SQL появились специальные типы данных: *date* и *time*. Но в таком подходе имеются недостатки: СУБД не знает семантики временного поля отношения и не может контролировать корректность его значений. В связи с этим появляется дополнительная избыточность хранения (предыдущее состояние объекта данных хранится и в основной БД, и в журнале изменений).

Существует отдельное направление исследований и разработок в области так называемых темпоральных БД. В этой области исследуются вопросы моделирования данных, языки запросов, организация данных во внешней памяти и т.д. Основной тезис темпоральных систем состоит в том, что для любого объекта данных, созданного в момент времени *t1* и уничтоженного в момент времени *t2*, в БД сохраняются (и доступны пользователям) все его состояния во временном интервале [*t1*,*t2*].

Исследования и построения прототипов темпоральных СУБД обычно выполняются на основе некоторой реляционной СУБД в виде *надстройки над реляционной системой*.

Контрольные вопросы

1. Назовите основные направления в совершенствовании реляционных баз данных.
3. В чем заключается метод генерации систем баз данных?
4. Перечислите способы оптимизации запросов.
5. Для каких задач применяются темпоральные запросы?

Глава 14

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ СУБД

14.1. Общие понятия объектно-ориентированного подхода

Направление объектно-ориентированных баз данных появлялось в середине 1980-х гг. Наиболее активно это направление развивается в последние годы.

Возникновение направления ООБД определяется прежде всего потребностями практики — необходимостью разработки сложных информационных прикладных систем, для которых технология предшествующих систем БД не была вполне удовлетворительной.

Конечно, ООБД возникли не на пустом месте. Соответствующий базис обеспечивают как предыдущие работы в области БД, так и давно развивающиеся направления языков программирования с абстрактными типами данных и объектно-ориентированных языков программирования.

Что касается связи с предыдущими работами в области БД, то, на наш взгляд, наиболее сильное влияние на работы в области ООБД оказывают проработки реляционных СУБД и следующее (хронологически) за ними семейство БД, в которых поддерживается управление сложными объектами. Кроме того, исключительное влияние на идеи и концепции ООБД и всего объектно-ориентированного подхода оказал подход к семантическому моделированию данных. Достаточное влияние оказывают также развивающиеся параллельно с ООБД направления дедуктивных и активных БД.

В наиболее общей постановке объектно-ориентированный подход базируется на концепциях:

- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом все время его существования и не меняется при изменении состояния объекта.

Каждый объект имеет состояние и поведение. Состояние объекта — набор значений его атрибутов. Поведение объекта — набор методов (программный код), оперирующих над состоянием объекта. Значение атрибута объекта — это тоже некоторый объект или мно-

жество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействуют объекты на основе передачи сообщений и выполнения соответствующих методов.

Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты-экземпляры которых не имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется *доменом* этого атрибута.

Допускается порождение нового класса на основе уже существующего класса — наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различаются случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, то набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.

Одной из более поздних идей объектно-ориентированного подхода является идея возможного переопределения атрибутов и методов суперкласса в подклассе (перегрузки методов). Эта возможность увеличивает гибкость, но порождает дополнительную проблему: при компиляции объектно-ориентированной программы могут быть неизвестны структура и программный код методов объекта, хотя его класс (в общем случае — суперкласс) известен. Для разрешения этой проблемы применяется так называемый метод позднего связывания, означающий интерпретационный режим выполнения программы с распознаванием деталей реализации объекта во время выполнения посылки сообщения к нему. Введение некоторых ограничений на способ определения подклассов позволяет добиться эффективной реализации без потребностей в интерпретации.

При таком наборе базовых понятий, если не принимать во внимание возможности наследования классов и соответствующие проблемы, объектно-ориентированный подход очень близок к подходу языков программирования с абстрактными (или произвольными) типами данных.

С другой стороны, если абстрагироваться от поведенческого аспекта объектов, объектно-ориентированный подход весьма бли-

зок к подходу семантического моделирования данных (даже по терминологии). Фундаментальные абстракции, лежащие в основе семантических моделей, неявно используются и в объектно-ориентированном подходе. На абстракции агрегации основывается построение сложных объектов, значениями атрибутов которых могут быть другие объекты. Абстракция группирования — основа формирования классов объектов. На абстракциях специализации (обобщения) основано построение иерархии или решетки классов.

Наиболее важным новым качеством ООБД, которое позволяет достичь объектно-ориентированного подхода, является поведенческий аспект объектов. В прикладных информационных системах, основывавшихся на БД с традиционной организацией (вплоть до тех, которые базировались на семантических моделях данных), существовал принципиальный разрыв между структурной и поведенческой частями. Структурная часть системы поддерживалась всем аппаратом БД, ее можно было моделировать, верифицировать, а поведенческая часть создавалась изолированно. В частности, отсутствовали формальный аппарат и системная поддержка совместного моделирования и гарантирования согласованности этих структурной (статической) и поведенческой (динамической) частей. В среде ООБД проектирование, разработка и сопровождение прикладной системы становятся процессом, в котором интегрируются структурный и поведенческий аспекты. Конечно, для этого нужны специальные языки, позволяющие определять объекты и создавать на их основе прикладную систему.

Специфика применения объектно-ориентированного подхода для организации и управления БД потребовала уточненного толкования классических концепций и некоторого их расширения. Это определилось потребностями долговременного хранения объектов во внешней памяти, ассоциативного доступа к объектам, обеспечения согласованного состояния ООБД в условиях мультидоступа и тому подобных возможностей, свойственных базам данных. Выделяют три аспекта, отсутствующих в традиционной парадигме, но требующихся в ООБД.

Первый аспект касается потребности в средствах спецификации знаний при определении класса (ограничений целостности, правил дедукции и т. п.).

Второй аспект — потребность в механизме определения разного рода семантических связей между объектами разных классов. Фактически это означает требование полного распространения на ООБД средств семантического моделирования данных. Потребность в использовании абстракции ассоциирования отмечается и в связи с использованием ООБД в сфере автоматизированного проектирования и инженерии.

Третий аспект связан с пересмотром понятия класса. В контексте ООБД оказывается более удобным рассматривать класс как множество объектов данного типа, т.е. одновременно поддерживать понятия и типа, и класса объектов.

14.2. Объектно-ориентированные модели данных

Первой формализованной и общепризнанной моделью данных была реляционная модель Кодда. В этой модели, как и во всех следующих, выделялись три аспекта: структурный, целостный и манипуляционный. Структуры данных в реляционной модели основываются на плоских нормализованных отношениях, ограничения целостности выражаются с помощью средств логики первого порядка, манипулирование данными осуществляется на основе реляционной алгебры или равносильного ей реляционного исчисления. Своим успехом реляционная модель данных во многом обязана тому, что она опирается на строгий математический аппарат реляционной алгебры и теории множеств

Основные трудности объектно-ориентированного моделирования данных связаны с тем, что не существует конкретного математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных. Разработка методов управления данными внутри объектов, как и любой процесс программирования нетрадиционных задач, остается «искусством программирования».

Методы могут быть *публичными* (доступными из объектов других классов) или *приватными* (доступными только внутри данного класса).

Итак, объектно-ориентированная система управления базами данных представляет собой объединение системы программирования и СУБД и основана на объектно-ориентированной модели данных.

Основное назначение ООБД связано с потребностью создания единого информационного пространства.

В этой среде должны отсутствовать противоречия между структурной и поведенческой частями проекта и должно поддерживаться эффективное управление сложными структурами данных во внешней памяти.

В отличие от традиционных реляционных систем, в которых при создании приложения приходится одновременно использовать процедурный язык программирования, ориентированный на работу со скалярными значениями, и декларативный язык запросов, ориентированный на работу с множествами, языковая среда ООБД — это объектно-ориентированная система программирования, естественно включающая в себя средства работы с долговременными объектами. Естественность включения средств работы с

БД в язык программирования означает, что работа с долговременными (хранимыми во внешней БД) объектами должна происходить на основе тех же синтаксических конструкций (и с той же семантикой), что и работа с временными объектами, существующими только во время работы программы.

Эта сторона ООБД наиболее близка родственному направлению языков программирования баз данных. Языки программирования ООБД и БД во многих своих чертах различаются только терминологически; существенным отличием является лишь поддержание в языках ООБД подхода к наследованию классов.

Другим аспектом языкового окружения ООБД является потребность в языках запросов, которые можно было бы использовать в интерактивном режиме. Если доступ к объектам внешней БД в языках программирования ООБД носит в основном навигационный характер, то для языков запросов более удобен декларативный стиль. Как известно, декларативные языки запросов менее развиты, чем языки программирования.

14.3. Языки программирования объектно-ориентированных баз данных

К настоящему моменту неизвестен какой-либо язык программирования ООБД, который был бы спроектирован целиком заново, начиная с нуля. Естественным подходом к построению такого языка было использование (с необходимыми расширениями) некоторого существующего объектно-ориентированного языка.

Одним из первых языков для реализации объектно-ориентированного и функционального подходов к программированию является язык Лисп (Common Lisp).

Потребности в еще более эффективной реализации заставляют в качестве основы объектно-ориентированного языка использовать известные языки программирования BASIC (Бейсик) и Си++.

Потребность в поддержании в объектно-ориентированной СУБД не только языка (или семейства языков) программирования ООБД, но и развитого языка запросов в настоящее время осознается практически всеми разработчиками. Система должна поддерживать легко осваиваемый интерфейс, прямо доступный конечному пользователю в интерактивном режиме.

Наиболее распространенный подход к организации интерактивных интерфейсов с объектно-ориентированными системами баз данных основывается на использовании так называемых обходчиков. В этом случае конечный интерфейс обычно является графическим. На экране отображается схема (или подсхема) ООБД, и пользователь осуществляет доступ к объектам в навигационном стиле. Некоторые исследователи считают, что в этом случае ра-

зумно игнорировать принцип инкапсуляции объектов и предъявлять пользователю внутренность объектов. В большинстве существующих систем ООБД подобный интерфейс существует, но всем понятно, что навигационный язык запросов — это в некотором смысле шаг назад по сравнению с языками запросов даже реляционных систем.

Неавигационные языки запросов. В настоящее время существует три подхода к разработке таких языков.

Первый подход — это расширение языков запросов реляционных систем. Наиболее распространены языки с синтаксисом, близким к языку SQL. Это связано, конечно, с общим признанием и чрезвычайно широким распространением этого языка.

Второй подход основывается на построении полного логического объектно-ориентированного языка исчисления.

Третий подход основывается на применении декларативного и объектно-ориентированного методов программирования.

Независимо от применяемого для разработки языка запросов подхода перед разработчиками встает одна концептуальная проблема, решение которой не укладывается в традиционное русло объектно-ориентированного подхода.

Исходя из концепции ООБД основой для формулирования запроса должен служить класс, представляющий в ООБД множество однотипных объектов. Но что может представлять собой результат запроса? Набор основных понятий объектно-ориентированного подхода не содержит подходящего к данному случаю понятия. Обычно из положения выходят, расширяя базовый набор концепций множества объектов и полагая, что результатом запроса является некоторое подмножество объектов — экземпляров класса. Это довольно ограниченный подход, поскольку автоматически исключается возможность наличия в языке запросов средств, аналогичных реляционному оператору соединения.

Проблемы оптимизации запросов. Основной целью оптимизации запроса в системе ООБД является создание оптимального плана выполнения запроса с использованием доступа к внешней памяти ООБД.

Оптимизация запросов хорошо исследована и разработана в контексте реляционных БД. Известны методы синтаксической и семантической оптимизации на уровне непроцедурного представления запроса, алгоритмы выполнения элементарных реляционных операций, методы оценок стоимости планов запросов.

Конечно, объекты могут иметь существенно более сложную структуру, чем кортежи плоских отношений, но не это различие является наиболее важным. Основная сложность оптимизации запросов к ООБД следует из того, что в этом случае условия выборки формулируются в терминах внешних атрибутов объектов (мето-

дов), а для реальной оптимизации (т.е. для выработки оптимального плана) требуются условия, определенные на внутренних атрибутах (переменных состояния).

Похожая ситуация существует и в реляционных СУБД при оптимизации запроса над представлением БД. В этом случае условия также формулируются в терминах внешних атрибутов (атрибутов представления), и в целях оптимизации запроса эти условия должны быть преобразованы в условия, определенные на атрибутах хранимых отношений. Хорошо известным методом такой предоптимизации является подстановка представлений, которая часто (хотя и не всегда в случае использования языка SQL) обеспечивает требуемые преобразования.

В системах ООБД ситуация существенно усложняется двумя обстоятельствами.

Во-первых, методы обычно программируются на некотором процедурном языке программирования и могут иметь параметры, т.е. в общем случае тело метода представляет не просто арифметическое выражение, как в случае определения атрибутов представления, а параметризованную программу, включающую в себя ветвления, вызовы функций и методов других объектов.

Во-вторых, точная реализация метода и даже структура объекта могут быть неизвестны во время компиляции запроса.

Одним из подходов к упрощению проблемы является открытие видимости некоторых (наиболее важных для оптимизации) внутренних атрибутов объектов. В этом контексте достаточно было бы открыть видимость только для компилятора запросов, т.е. фактически запретить переопределять такие переменные в подклассах. С точки зрения пользователя такие атрибуты выглядели бы как методы без параметров, возвращающие значение соответствующего типа. Однако лучше было бы сохранить строгую инкапсуляцию объектов (чтобы избавить приложение от критической зависимости от реализации) и обеспечить возможности тщательного проектирования схемы ООБД с учетом потребностей оптимизации запросов.

Общий подход к предоптимизации условия выборки для одного класса объектов может быть следующим.

1. Преобразовать логическую формулу условия в конъюнктивную нормальную форму. Мы не останавливаемся на способе выбора конкретной КНФ, но, естественно, должна быть выбрана хорошая КНФ (например, содержащая максимальное число атомарных конъюнктов).

2. Для каждого конъюнкта, включающего в себя методы с известным во время компиляции телом, заменить вызовы методов на их тела с подставленными параметрами. (Будем предполагать, что параметры не содержат вызовов функций или методов других объектов.)

3. Для каждого такого конъюнкта произвести все возможные упрощения, т.е. вычислить все, что можно вычислить в статике. Хотя в общем виде эта задача является очень сложной, при разумном проектировании ООБД в число методов должны будут войти методы с предельно простой реализацией, задавать условия на которых будет очень естественно. Такие условия будут упрощаться очень эффективно.

4. Если появились конъюнкты, представляющие собой простые предикаты сравнения на основе переменных состояния и констант, использовать эти конъюнкты для выработки оптимального плана выполнения запроса. Если же такие конъюнкты получить не удалось, единственным способом «отфильтровать» класс объектов является его последовательный просмотр с полным вычислением (возможно упрощенного) логического выражения для каждого объекта.

Возможности оптимизации будут зависеть от особенностей языка программирования, который используется для программирования методов, особенностей конкретного языка запросов и от того, насколько продуманно спроектирована схема ООБД. В частности, желательно, чтобы используемый язык программирования стимулировал максимально дисциплинированный стиль программирования методов объектов. Язык запросов должен разумно ограничивать возможности пользователей (в частности, в отношении параметров методов, участвующих в условиях запросов). В классах схемы ООБД должны содержаться простые методы, не переопределяемые в подклассах и основанные на тех переменных состояния, которые служат основой для организации методов доступа.

Указанные ограничения не влекут зависимости прикладной программы от особенностей реализации ООБД, поскольку объекты остаются полностью инкапсулированными. Использование в условиях запросов простых методов должно стимулироваться не требованиями реализации, а семантикой объектов.

Контрольные вопросы

1. Назовите принципы объектно-ориентированного подхода к созданию баз данных.
2. Какие объектно-ориентированные модели данных вы знаете?
3. Какие языки программирования применяют для разработки объектно-ориентированных баз данных?

Глава 15

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ СУБД CACHE

15.1. Структура СУБД

СУБД Cache относится к постреляционным объектно-ориентированным СУБД. Термин «постреляционная СУБД» обозначает принадлежность к СУБД нового поколения. Имеется в виду не столько аспект времени (СУБД появилась после своих основных реляционных конкурентов), сколько ряд технологических новшеств, таких как единая архитектура данных и полная поддержка объектно-ориентированных технологий.

В соответствии с изложенными ранее принципами проектирования объектно-ориентированных баз данных система Cache имеет следующие возможности:

- наличие объектов — элементов БД, в которых хранятся не только данные, но и методы их обработки;
- система позволяет обрабатывать мультимедийные данные и предоставляет пользователям возможность создавать собственные структуры данных любой сложности;
- ООБД допускают работу на высоком уровне абстракции.

Отличительной особенностью СУБД является независимость хранения данных от способа их представления, что реализуется с помощью так называемой единой архитектуры данных. В рамках данной архитектуры существует единое описание объектов и таблиц, отображаемых непосредственно в многомерные структуры ядра базы данных, ориентированных на обработку транзакций. Как только определяется класс объектов, автоматически генерируется реляционное описание данных этого класса в формате SQL. Подобным же образом, как только в Словарь данных поступает язык определения данных (ЯОД) — описание в формате реляционной базы данных, автоматически генерируется реляционное и объектное описание данных, тем самым устанавливается доступ в формате объектов. При этом все описания ведутся согласованно, все операции по редактированию проводятся только с одним описанием данных. Это позволяет сократить время разработки, сэкономить вычислительные ресурсы. Приложения будут работать значительно быстрее. В табл. 15.1 представлены основные элементы архитектуры СУБД:

- платформы, на которых работает система;
- многомерный сервер данных;
- три способа доступа к данным;

Таблица 15.1

Архитектура постреляционной СУБД Cache

Direct	Objects	SQL	WEB
Cache Object Script			
MD	Objects	SQL	
MDS			
Платформы			

- язык описания бизнес-логики Cache' ObjectScript;
- интерфейсы к средствам проектирования и разработки приложений и Web-технология Cache' Server Pages.

Cache — многоплатформенная система. Cache поддерживает следующие операционные системы: всю гамму ОС Windows, Linux, основные реализации Unix и Open VMS. Планируется поддержка новых реализаций Unix. Большое внимание уделяется новой платформе Itanium.

Данные в Cache хранятся под управлением многомерного сервера данных (MDS). В основе Cache лежит транзакционная многомерная модель данных, которая позволяет хранить и представлять данные так, как они чаще всего используются. Многомерный сервер данных снимает многие ограничения, накладываемые реляционными СУБД, которые хранят данные в двумерных таблицах. Как известно, реляционная модель БД состоит из большого числа таблиц, что необходимо при работе со сложными структурами данных. Это, в свою очередь, существенно усложняет и замедляет выполнение сложных транзакций и ведет к хранению излишней информации. Cache хранит данные в виде многомерных разреженных массивов.

Уникальная транзакционная многомерная модель данных позволяет избежать проблем, присущих реляционным СУБД, оптимизируя данные уже на уровне хранения.

Многомерный сервер данных Cache предназначен для обработки транзакций в системах с большими и сверхбольшими базами данных (сотни гигабайт, терабайты) и большим числом одновременно работающих пользователей. Многомерный сервер данных Cache позволяет получить высокую производительность, отказавшись от хранения избыточных данных и таблиц.

Транзакционная модель данных Cache позволяет оптимизировать данные на уровне хранения, поддерживать объектную модель и сложные типы данных. Все эти возможности значительно упрощают создание сложных систем.

В Cache реализована концепция единой архитектуры данных, т.е. к одним и тем же данным, хранящимся под управлением мно-

гомерного сервера данных Cache есть три способа доступа: прямой, реляционный и объектный.

Прямой доступ к данным (*Cache Direct Access*) обеспечивает максимальную производительность и полный контроль со стороны программиста. Разработчики приложений получают возможность работать напрямую со структурами хранения. Использование этого типа доступа накладывает определенные требования на квалификацию разработчиков, позволяет оптимизировать хранение данных приложения и создавать сверхбыстрые алгоритмы обработки данных.

Реляционный доступ к данным (*Cache SQL*) обеспечивает максимальную производительность реляционных приложений с использованием встроенного языка SQL. Cache SQL соответствует стандарту SQL 92. Кроме того, разработчик может использовать разные типы триггеров и хранимых процедур.

Даже без использования прямого и объектного доступа к данным приложения на Cache работают быстрее за счет высокой производительности многомерного сервера данных.

Объектный доступ к данным (*Cache Objects*) осуществляется при использовании объектно-ориентированных языков программирования Java, Visual C++, VB и других ActiveX-совместимых средств разработки, таких как PowerBuilder и Delphi. Для этого в Cache реализована объектная модель управления базами данных, в которой полностью поддерживаются наследование (в том числе и множественное), инкапсуляция и полиморфизм. При создании информационной системы разработчик получает возможность использовать объектно-ориентированный подход к разработке, моделируя предметную область в виде совокупности классов объектов, в которых хранятся данные (свойства классов) и поведение классов (методы классов).

Cache, поддерживая объектную модель данных, позволяет естественным образом использовать объектно-ориентированный подход как при проектировании предметной области, так и при реализации приложений средствами разработки (Java, C++, Delphi, VB).

Как только определяется класс объектов, Cache автоматически генерирует реляционное описание этих данных так, что к ним можно обращаться, используя SQL.

Подобным же образом при импорте в Словарь данных (описаний реляционной базы данных) Cache автоматически генерирует реляционное и объектное описание данных, открывая тем самым доступ к данным, как к объектам. При этом все описания данных ведутся согласованно, все операции по редактированию проводятся только над одним экземпляром данных. Кроме того, программист может обратиться к тем же данным с помощью прямого доступа.

Cache позволяет комбинировать тремя типами доступа к данным, оставляя при этом разработчику свободу выбора. Например, при реализации системы объектный доступ может использоваться при описании бизнес-логики приложения и создании пользовательского интерфейса с помощью объектно-ориентированных средств разработки (VB, Delphi, C++).

Реляционный доступ может применяться для совместимости с другими системами и интеграции с инструментами построения отчетов и аналитической обработки данных (Seagate Info, Cognos, Business Objects).

Прямой доступ обеспечивает максимальную производительность и может быть использован для реализации таких операций, в которых применение обычных хранимых процедур, основанных на SQL, не может обеспечить необходимую производительность. Использование прямого доступа для реализации сложных операций позволяет увеличить производительность на один-два порядка.

Для реализации бизнес-логики в СУБД используется специальный язык Cache Object Script (COS) — полнофункциональный язык, который имеет все необходимые механизмы для работы с данными с помощью любого способа доступа. С помощью COS разработчик создает методы классов, триггеры, хранимые процедуры, различные служебные программы. Стоит отметить интерфейсы со средствами проектирования и разработки приложений. Специальные компоненты Cache позволяют проектировать приложения как для объектного, так и для реляционного методов обработки данных.

Кроме того, поддерживаются следующие интерфейсы: Native C++, Java, EJB, ActiveX, XML, а также интерфейсы CallIn и CallOut.

Для обеспечения надежности в Cache предусмотрены такие механизмы, как журнал до и после записи, теневой сервер, репликация, «горячее» резервное копирование и т.д.

Протокол распределенного кэша (Cache Distibuted Cache Protocol) — уникальная сетевая технологи фирмы InterSystems, которая распределяет базу данных по сети, оптимизируя производительность и пропускную способность сети в зависимости от работы приложений.

Cache — открытая система, в которой поддерживается множество интерфейсов к средствам проектирования и разработке приложений.

Cache работает практически на всех популярных платформах с наиболее распространенными Web-серверами. При этом обеспечивается полная переносимость приложений с платформы на платформу.

При всех достоинствах современной объектно-ориентированной технологии разработки баз данных имеется несколько препятствий,

которые удерживают разработчиков от принятия решения о переходе с реляционной технологии на объектно-ориентированную. Основным препятствием является значительный объем разработок, опирающихся на реляционные СУБД. При переходе на новую технологию обработки данных необходимо многое начинать «с нуля», поэтому возникает вопрос целесообразности такого перехода.

Кроме того, объектная технология, поддерживаемая в ряде постреляционных СУБД, не имеет развитого и стандартизированного языка генерации отчетов и анализа данных, каким является структурированный язык запросов SQL.

В системе Cache данные проблемы были решены за счет процедур, обеспечивающих возможность перехода с реляционной технологии на объектную.

15.2. Cache и WWW-технологии

В условиях разработки и внедрения на предприятиях CALS-технологий Web-сервисы являются одной из необходимых возможностей, которые предоставляет среда Cache.

Как известно, основной принцип работы WWW состоит в том, что все документы создаются в едином формате HTML, а клиентские приложения-браузеры, функционирующие на самых различных компьютерных платформах, почти одинаково отображают эти документы. Однако решения отражают статические принципы обработки документов. Для придания им динамики требовалось использование различных средств программирования, действующих как на стороне клиента, так и на стороне сервера.

Если разработчик применял скриптовые языки, функционирующие на стороне клиента, то область их действия была ограничена отображаемым документом, а в качестве входных данных использовались либо действия пользователя, либо информация, хранящаяся на его компьютере. Это ограничение связывалось с тем, что скриптовые языки не получали какой-либо дополнительной информации от WWW-сервера, с которого был загружен документ, вместе с которым и были получены скрипты.

Если на сайте требовалось использовать несколько более сложные способы управления данными, то приходилось применять исполняемые модули, функционирующие на самом сервере. Только таким образом можно было осуществлять доступ к базам данных и на основе полученной информации формировать страницы, передаваемые затем удаленному пользователю. Только на сервере удавалось обеспечить поддержку сеансов работы и идентификацию пользователей, столь необходимые во всех приложениях электронной коммерции.

Естественно, технологий Web-программирования было немало. Для этой цели использовались и классические языки, такие как C++, и технологии, специально ориентированные на Web, такие как ASP. Однако все они имели одну общую черту — конечный вывод документов производился в формате HTML.

Если учитывать общую скорость прогресса в компьютерной индустрии, то можно признать, что долгожитель HTML, который достаточно долго использовался в сети Интернет, может в ближайшем будущем быть заменен новым языком XML. Более того, международные стандарты по CALS уже ориентируются на этот язык форматирования документов.

Система Web-сервисов и обособленных клиентов нужна только для ресурсов с чрезвычайно сложной функциональностью.

В общем случае Web-сервисы принимают и передают информацию на языке XML. Этот язык, естественно, является открытым стандартом. Также следует учитывать, что XML, как и его предшественник HTML, не зависит от платформы и операционной системы. Сочетание этих двух факторов позволяет разработчикам свободно создавать самые различные клиентские приложения, функционирующие на разных компьютерных платформах, т.е. одному сервису может соответствовать несколько клиентских приложений.

XML-документы, пересылающиеся от сервера к клиенту и обратно, передаются по протоколу HTTP, который пропускается практически всеми брандмауэрами, что также прибавляет привлекательности идее Web-сервисов.

Следует отметить и тот факт, что Web-сервисы являются самодокументируемыми, т.е. любое клиентское приложение может получить информацию о структуре сервиса, его функциональности и правилах вызова функций, поддерживаемых Web-сервисом.

Web-сервисы способны передавать и получать информацию тремя способами: с применением методов GET и POST стандартного протокола HTTP или при помощи языка SOAP, который является производным от XML. Первые два варианта разрешают использовать в качестве клиентского приложения стандартный браузер, но необходимо отдавать себе отчет, что это далеко не идеальный вариант. Во-первых, при помощи браузера весьма трудно организовать вызов всех функций достаточно сложного сервиса. Разработчику необходимо исследовать структуру сервиса заранее, перед тем как создавать Web-страницы для доступа к сервису. Так что в этом случае мы теряем преимущества самодокументируемости. Во-вторых, следует помнить, что вывод информации все равно будет идти в «чистом» XML, который браузер не сможет адекватно отобразить. Поэтому для работы с Web-сервисами, обладающими достаточно серьезной функциональностью, удобно использовать язык SOAP.

15.3. Visual Basic.NET — среда разработки приложений

Visual Basic.NET — современная визуальная среда быстрой разработки Windows-приложений, создания полнофункциональных профессиональных проектов на базе современной объектно-орентированой концепции. Конструирование пользовательского интерфейса в Visual Basic.NET сведено к работе с мышью. Иногда требуется лишь добавить несколько строк кода, созданного мастером проекта. Visual Basic.NET — это достаточно простой, строго типизированный язык, с помощью которого легко строить объектно-орентированные конструкции, проектировать графический пользовательский интерфейс (GUI), работать с базами данных.

Таким образом, Visual Basic.NET, интегрированный с Cache, обеспечивает:

- простоту создания пользовательского интерфейса программы;
- возможность работы с Web-сервисами;
- создание клиент-серверных приложений, включая работу через Интернет;
- поддержку SOAP-протокола.

15.4. SOAP — многоплатформенный протокол передачи данных

SOAP-протокол — это набор правил для приложений, которые могут вызывать методы удаленных объектов. Где именно находятся эти удаленные объекты — в другом каталоге, где-то в корпоративной интрасети или в Интернете — для клиентских программ, использующих SOAP, абсолютно неважно. SOAP основан на языке XML. Каждая передача информации между клиентом и сервером является отдельным XML-документом, который написан по правилам SOAP.

Как известно, логическая структура каждого XML-документа определяется его DTD-блоком. Для SOAP заранее определены все возможные теги и типы данных, поэтому SOAP-документы не нуждаются в DTD-блоках. За счет подобной унификации серверы и клиенты освобождаются от достаточно тяжелой процедуры синтаксического анализа приходящего документа с не определенным заранее набором тегов.

Протокол SOAP — это слабосвязанный механизм, ориентированный на сообщения и предназначенный для удаленного вызова объектов по глобальным сетям.

Рассмотрим принцип его работы. Зная URL, клиент на языке описания сервиса SDL (Service Description Language) запрашивает у сервера информацию о нем (предоставляемые методы, параметры и т.д.). В ответ клиент получает SDL-файл с нужными сведениями о том, какие услуги (методы) ему будут предоставляться и как ими пользоваться.

Методы вызываются с помощью HTTP-запросов и ответов. В запрос вкладывается XML-текст, состоящий из трех частей:

- SOAP envelope (пакет), определяющий содержимое сообщения;
- заголовок SOAP, определяющий возможные заголовки сообщения;
- тело SOAP, содержащее информацию о запросе или ответе на запрос.

Ответ приходит также в виде XML, содержащего результаты обработки запроса или код ошибки.

Новая версия Cache — Cache 5 — реализует протокол SOAP для всех поддерживаемых платформ. При этом не требуется использование промежуточных приложений сторонних компаний, все необходимые функции реализованы в виде системных классов БД Cache.

Cache SOAP предоставляет широкий спектр функций, который включает в себя:

- создание Web-служб путем определения классов, содержащих Web-методы, предоставляемые удаленным системам. При этом методы вызываются непосредственно в среде БД, что приводит к увеличению производительности за счет сокращения трафика между клиентом и сервером;
- автоматическое создание и публикацию каталога доступных методов (WSDL).

Для создания Web-службы и публикации каталога на сервере не требуется никаких дополнительных знаний о Cache. Реализация служб осуществляется на основе объектной модели Cache. При этом для создания службы необходимо определить класс, наследуемый от системного класса %SOAP.WebService. Методы, которые подлежат публикации на сервере, характеризуются параметром WebMethod. Код, реализующий логику метода, как и код обыкновенного метода класса Cache, может быть реализован с использованием Cache Object Script, Cache Basic или Cache SQL. Возможно использование сложных типов данных (встраиваемые объекты, коллекции, отношения и др.) в качестве аргументов и возвращаемого значения публикуемого метода. При этом Cache автоматически создает необходимые объекты сложных типов данных при поступлении соответствующего запроса на сервер.

Приведем пример Web-службы с одним методом:

```
Class MyApp.StockService Extends %SOAP.WebService
{
Parameter SERVICENAME = "MyStockService";
Parameter LOCATION = "http://localhost/csp/user";
Parameter NAMESPACE = "http://tempuri.org";
/// Метод возвращает цену на завтра
```

```
ClassMethod Forecast (StockName As %String) As %Integer
[WebMethod]
{
// применяем генератор случайных чисел
Set price = $Random(1000)
Quit price
}
}
```

Кроме того, Cache автоматически генерирует тестовые CSP-страницы методов и CSP-страницу каталога, которые можно использовать для проверки правильной работы создаваемой службы. При этом CSP-страницы представляются в виде обыкновенных Web-приложений, доступных для просмотра обычным браузером.

Cache SOAP Server работает следующим образом:

- для каждого Web Service создается новый Cache-класс, который расширяется с помощью %S SOAP.WebService$;
- с помощью этого класса определяется один или более методов, которые соответствуют методам Web Service. Каждый из них может быть определен как WebMethod при добавлении ключевого слова «WebMethod» в его описании. Можно также определить Web-методы, которые возвращают массивы объектов, с помощью объявления запроса и добавления ключевого слова «WebMethod» в его описании;
- компилируется Web Service класс, при этом компилятор Cache автоматически привязывает каталог с информацией, описывающей содержимое SOAP-сервиса и строит SOAP-интерфейс для каждого Web-метода. SOAP-интерфейс для Web-метода является сгенерированным классом, который выполняет преобразование запроса SOAP в специфический вызов Web-метода, используя Cache XML-технологию;
- затем Cache автоматически создает WSDL-документ для каждого Web-сервиса. WSDL-документ является XML-документом, предоставляющим список доступных методов, сигнатуру и детали о том, как они могут быть вызваны с помощью SOAP-клиента. WSDL-документ создается с помощью Web (HTTP)-сервера, использующего CSP. WSDL-документ является динамически создаваемым и будет автоматически отображать любые изменения Web Service-класса;
- Soap-клиент открывает доступный Web-сервис, запрашивая WSDL-документ, который, в свою очередь, посылает запрос Cache-серверу. Используя эту информацию в WSDL-документе, SOAP-клиент активизирует определенный метод с помощью создания XML-сообщения и посылки его (с помощью HTTP) SOAP-серверу;
- Cache SOAP-сервер получает вызов SOAP с помощью Cache (CSP) HTTP Gateway. Сервер распаковывает сообщение, прове-

ряет его правильность и вызывает определенный Web-метод. Перед вызовом Web-метода Cache SOAP-сервер конвертирует все входные параметры к соответствующему представлению Cache;

• Web-метод выполняет свой код и возвращает ответ. Этот ответ может быть простой строковой константой, может быть XML-объектом или массивом.

Cache обеспечивает возможность создания SOAP client — класса, содержащего методы, который вызывает Web-сервис, используя протокол SOAP.

Cache — пользователь SOAP работает следующим образом:

• для каждого Web-service (набор связанных методов SOAP), которые вы желаете вызвать, необходимо создать новое определение класса Cache, которое получено от %SOAP.WebClient, находящегося в библиотеке Cache;

• класс пользователя SOAP содержит один или более методов класса, которые соответствуют методам Web-service. Каждый из этих методов пользователя определен как являющийся WebMethod с помощью добавления ключевого слова «WebMethod» в его описании;

• при компиляции класса — пользователя SOAP транслятор класса Cache автоматически транслирует информацию каталога, описание содержания SOAP, создает интерфейс пользователя SOAP для каждого Web-service;

• интерфейс пользователя SOAP для Web-service — сгенерированный класс, который исполняет работу преобразования запроса SOAP в определенный запрос Web-service, используя технологию перевода объекта в формат XML;

• SOAP Client обнаруживает доступный Web-service с помощью запроса WSDL-документа от Web-server, который, в свою очередь, запрашивает это от сервера Cache. Используя эту информацию в WSDL-документе, SOAP Client вызывает определенный метод, создавая XML-сообщение и отправляя его (через HTTP) на сервер SOAP, как определено в WSDL-документе;

• сервер SOAP получает запрос SOAP, распаковывает сообщение, проверяет его и вызывает указанную операцию.

Контрольные вопросы

1. Чем принципиально отличается СУБД Cache от реляционных СУБД?
2. Как осуществляется в системе прямой доступ к данным?
3. Какие протоколы передачи информации в сети Интернет поддерживаются в Cache?
4. Перечислите особенности среды разработки приложений Visual Basic.NET.

Глава 16

СИСТЕМЫ БАЗ ДАННЫХ, ОСНОВАННЫЕ НА ПРАВИЛАХ

16.1. Структура базы данных

В базе данных хранится три вида информации.

1. Информация, характеризующая структуры пользовательских данных (описание структурной части схемы базы данных). Такая информация в случае реляционной базы данных сохраняется в системных отношениях-каталогах и содержит главным образом имена базовых отношений, имена и типы данных их атрибутов.

2. Наборы кортежей пользовательских данных, сохраняемых в определенных пользователями отношениях.

3. Правила, определяющие ограничения целостности базы данных, триггеры базы данных и представляемые (виртуальные) отношения.

В реляционных системах правила также сохраняются в системных таблицах-каталогах, хотя плоские таблицы далеко не идеально подходят для этой цели.

Информация первого и второго вида в совокупности явно описывает объекты (сущности) реального мира, моделируемые в базе данных. Другими словами, это явные факты, предоставленные пользователями для хранения в БД. Эту часть базы данных принято называть экстенсиональной.

Информация третьего вида служит для руководства СУБД при выполнении различного рода операций, задаваемых пользователями.

Ограничения целостности могут блокировать выполнение операций обновления базы данных, триггеры вызывают автоматическое выполнение специфицированных действий при возникновении специфицированных условий, определения представлений вызывают явную или косвенную материализацию представляемых таблиц при их использовании. Эту часть базы данных принято называть интенсиональной; она содержит не непосредственные факты, а информацию, характеризующую семантику предметной области.

Как видно, в реляционных базах данных наиболее важное значение имеет экстенсиональная часть, а интенсиональная часть играет в основном вспомогательную роль.

В системах баз данных, основанных на правилах, эти две части, как минимум, равноправны.

16.2. Активные базы данных

БД называется активной, если СУБД по отношению к ней выполняет не только те действия, которые явно указывает пользователь, но и дополнительные действия в соответствии с правилами, заложенными в саму БД.

Основа этой идеи содержалась в языке SQL. Действительно, что есть определение триггера или условного воздействия, как не введение в БД правила, в соответствии с которым СУБД должна производить дополнительные действия? Плохо лишь то, что на самом деле триггеры не были полностью реализованы ни в одной из известных систем. И это не случайно, потому что реализация такого аппарата в СУБД очень сложна, накладна и не полностью понятна.

Среди вопросов, ответы на которые до сих пор не получены, следующие.

Как эффективно определить набор вспомогательных действий, вызываемых прямым действием пользователя?

Каким образом распознавать циклы в цепочке действие — условие — действие — ... и что делать при возникновении таких циклов?

В рамках какой транзакции выполнять дополнительные условные действия и к бюджету какого пользователя относить возникающие накладные расходы?

Масса проблем не решена даже для сравнительно простого случая реализации триггеров SQL, а задача ставится уже гораздо шире. Предлагается иметь в составе СУБД продукционную систему общего вида, условия и действия которой не ограничиваются содержимым БД или прямыми действиями над ней со стороны пользователя. В условие может входить время суток, а действие может быть внешним, например вывод информации на экран оператора. Практически все современные работы по активным БД связаны с проблемой эффективной реализации такой продукционной системы.

Намного важнее в практических целях реализовать в реляционных СУБД аппарат триггеров. В проекте стандарта SQL3 предусматривается существование языковых средств определения условных воздействий. Их реализация и будет первым практическим шагом к активным БД (уже появились соответствующие коммерческие реализации).

16.3. Дедуктивные базы данных

Дедуктивная БД состоит из двух частей: экстенциональной, содержащей факты, и интенциональной, содержащей правила для логического вывода новых фактов на основе экстенциональной части и запроса пользователя.

При таком общем определении SQL-ориентированную реляционную СУБД можно отнести к дедуктивным системам. Действительно, что есть определенные в схеме реляционной БД представления, как не интенциональная часть БД. Не так уж важно, какой конкретный механизм используется для вывода новых фактов на основе существующих. В случае SQL основным элементом определения представления является оператор выборки языка SQL, что вполне естественно, поскольку результатом оператора выборки является порождаемая таблица. Обеспечивается и необходимая расширяемость, поскольку представления могут определяться не только над базовыми таблицами, но и над представлениями.

Основным отличием реальной дедуктивной СУБД от реляционной является то, что и правила интенциональной части БД, и запросы пользователей могут содержать рекурсию. Можно спорить о том, всегда ли хороша рекурсия. С одной стороны, возможность определения рекурсивных правил и запросов дает возможность простого решения в дедуктивных базах данных проблем, которые вызывают большие проблемы в реляционных системах (например, проблемы разборки сложной детали на примитивные составляющие). С другой стороны, именно возможность рекурсии делает реализацию дедуктивной СУБД очень сложной и во многих случаях неразрешимой проблемой.

Мы не будем более подробно рассматривать конкретные проблемы, применяемые ограничения и используемые методы в дедуктивных системах. Отметим лишь, что обычно языки запросов и определения интенциональной части БД являются логическими (поэтому дедуктивные БД часто называют логическими). Имеется прямая связь дедуктивных БД с базами знаний (интенциональную часть БД можно рассматривать как базу знаний (БЗ)). Более того, трудно провести грань между этими двумя сущностями; по крайней мере, общего мнения по этому поводу не существует.

Какова же связь дедуктивных БД с реляционными СУБД, кроме того, что реляционная БД является вырожденным частным случаем дедуктивной? Связь заключается в том, что для реализации дедуктивной СУБД обычно применяется реляционная система. Такая система выступает в роли хранителя фактов и исполнителя запросов, поступающих с уровня дедуктивной СУБД. Такое использование реляционных СУБД резко актуализирует задачу глобальной оптимизации запросов.

При обычном применении реляционной СУБД запросы обычно поступают на обработку по одному, поэтому нет повода для их глобальной (межзапросной) оптимизации. Дедуктивная же СУБД при выполнении одного запроса пользователя в общем случае генерирует пакет запросов к реляционной СУБД, которые могут оптимизироваться совместно.

Конечно, в случае когда набор правил дедуктивной БД становится большим и их невозможно разместить в оперативной памяти, возникает проблема управления их хранением и доступом к ним во внешней памяти. Здесь также может быть применена реляционная система, но уже не очень эффективно. Требуются более сложные структуры данных и другие условия выборки. Известны лишь частные попытки решить эту проблему, но общего решения пока нет.

Контрольные вопросы

1. Чем отличаются структуры таблиц базы данных, основанных на правилах, от традиционных — реляционных БД?
2. Назовите основные характеристики активных и дедуктивных баз данных.

ЧАСТЬ V

ПРАКТИЧЕСКИЕ ПРИМЕРЫ ПРИМЕНЕНИЯ СУБД В ПРОИЗВОДСТВЕ И БИЗНЕСЕ

Глава 17

СИСТЕМЫ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ПРОДУКЦИИ

17.1. Интегрированная информационная среда предприятия

В подразд. 4.1 мы уже рассматривали новое направление развития информационных систем в условиях современного производства и бизнеса — CALS-технологии и необходимость в связи с этим перехода к организации на предприятиях распределенных, многопользовательских БД. Рассмотрим основные направления организации работ по созданию на любом предприятии единого информационного пространства.

Как следует из концептуальной модели этого направления работ, основой, ядром CALS-технологий и создаваемых на этой основе автоматизированных систем является интегрированная информационная среда.

Представление об ИИС было введено в научный обиход задолго до появления CALS-технологий. Еще в 1983 г. японский ученый Н. Окино опубликовал работу, в которой утверждал, что производство материальных объектов и сопутствующие ему процессы проектирования, технологической подготовки и управления так сильно отличаются от других видов деятельности человека, что им должна отвечать особая архитектура программно-методического, математического и информационного обеспечения. По его мнению, принципиальная разница между обработкой информации в производственной системе и в других случаях применения вычислительной техники в основном сводится к двум положениям.

1. Производство и все процессы в нем принадлежат физическому миру, а процессы, протекающие в компьютере, — миру информации. Следовательно, необходимо преобразование производственных проблем в информационные, а также обратный переход из информационного мира в физический. Это проблема адекватного моделирования, т.е. установления соответствия (по возможности, взаимно-однозначного) между физическим и информационным пространством. При создании традиционного математи-

ческого обеспечения (МО) для решения вычислительных задач в центр разработки ставится единственная математическая модель проблемы, которая через прикладной интерфейс адаптируется к различным областям применения (рис. 17.1).

Такой подход к решению производственных проблем практически не реализуем, поскольку ввиду их сложности и многообразия единую модель создать невозможно.

Если в добавление к изучавшимся Н.Окино производственным проблемам включить в рассмотрение еще и проблемы поставок, эксплуатации, обслуживания и ремонта изделий, т.е. все постпроизводственные стадии жизненного цикла (ЖЦ) изделия, то ситуация становится еще более сложной.

2. В связи с отмеченными выше недостатками традиционного подхода (см. рис. 17.1) предлагается отбросить стратегию единственной модели организации баз данных и перейти к стратегии, сущность которой показана на рис. 17.2.

Здесь роль ядра системы играет не модель, а общая (интегрированная) база данных (ОБД), к которой могут обращаться различные проблемно-ориентированные модели, реализованные в форме программных приложений. Предполагается, что в ОБД хранятся информационные объекты (ИО), адекватно отображающие в информационном мире сущности физического мира, предметы, материалы, изделия, процессы и технологии, разнообразные документы, финансовые ресурсы, персонал подразделения и оборудование предприятия-изготовителя, условия эксплуатации изделия у заказчика, сервисной и ремонтной служб и т.д.

Упомянутые выше приложения обращаются в общую базу данных, находят в ней необходимые информационные объекты, об-

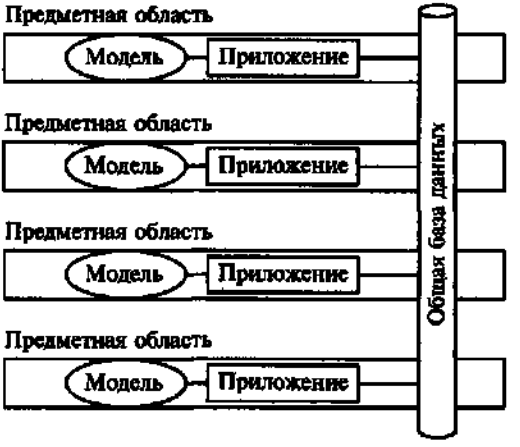


Рис. 17.1. Стратегия организации общей базы данных на основе локальных моделей

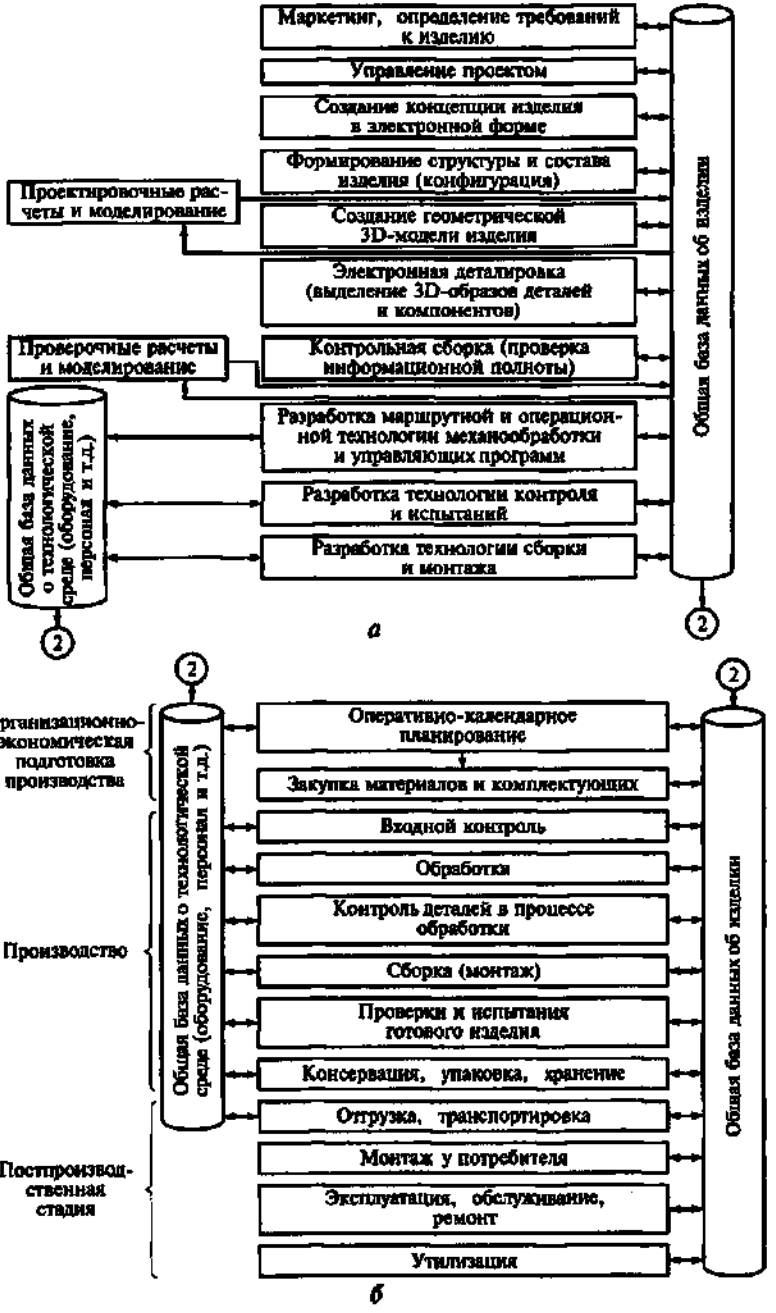


Рис. 17.2. Стратегия организации общей базы данных на основе многопользовательских (*а*) и распределенных (*б*) баз данных

рабатывают их и снова помещают в ОБД результаты этой обработки.

В какой-то мере Н. Окино предвосхитил появление объектно-ориентированного подхода, предложив рассматривать все, что происходит в информационном мире, на основе дуализма объект—операция.

Суть развиваемых Н. Окино идей состоит в следующем.

Любой сущности физического мира соответствует информационный объект, представляющий собой некоторый набор данных. Любой вид использования физической сущности, ее преобразование в другую сущность (или в ту же сущность, но с иными значениями параметров) — обработка, изготовление, измерение, проектирование — в информационном мире отображается операцией (командой, программой и т.д.).

Дальнейшее развитие информационных технологий привело к появлению объектно-ориентированного подхода, который позволил адекватно перевести многие процессы, протекающие на предприятии, в виртуальное информационное пространство, что и сделало актуальной всю проблематику, связанную с использованием CALS-технологий.

Сказанное относится, в частности, к процессам конструкторской и технологической подготовки производства, в ходе которых создается техническая документация различных видов и назначения, к процессам управления производством и бизнесом на всех уровнях, в которых по необходимости приходится иметь дело с большими объемами разнообразной информации. Сегодня эти процессы в значительной мере состоят из операций создания, преобразования, транспортировки и хранения информационных объектов в рамках интегрированной информационной среды.

17.2. Структура и состав интегрированной информационной среды предприятия

ИИС представляет собой хранилище данных, содержащее все сведения, создаваемые и используемые всеми подразделениями и службами предприятия — участниками жизненного цикла изделия в процессе их производственной деятельности. Это хранилище имеет сложную структуру, отражающую внешние и внутренние связи. ИИС должна включать в свой состав, как минимум, две основные базы данных:

- общую базу данных об изделии (изделиях) (ОБДИ);
- общую базу данных о предприятии (ОБДП).

На рис. 17.2, где представлена структура ИИС во взаимодействии с процессами, протекающими на протяжении всего жизненного цикла продукции, видно, что в этих процессах используется информация, содержащаяся в ИИС, а информационные

объекты, порождаемые в ходе процессов, возвращаются в ИИС для хранения и последующего использования в других процессах.

С общей базой данных об изделии связаны процессы на всех стадиях его жизненного цикла. Общая база данных о предприятии информационно связана с технологической и организационно-экономической подготовкой производства и собственно производством (включая процессы отгрузки и транспортировки готовой продукции).

При создании нового изделия и технологической подготовке его производства средствами конструкторских и технологических САПР (CAE/CAD/CAM) в ИИС создаются информационные объекты, описывающие структуру изделия, его состав и все входящие компоненты: детали, узлы, агрегаты, комплектующие, материалы и т.д. Каждый информационный объект обладает атрибутами, описывающими свойства физического объекта: технические требования и условия, геометрические (размерные) параметры, массогабаритные показатели, характеристики прочности, надежности, ресурса и другие свойства изделия и его компонентов.

Информационные объекты в составе общей базы данных содержат в произвольном формате информацию, требуемую для выпуска и поддержки технической документации, необходимой на всех стадиях производственного процесса для всех изделий, выпускаемых предприятием. Каждый информационный объект идентифицируется уникальным кодом и может быть извлечен из базы данных для выполнения действий с ним. Общая база данных об изделиях обеспечивает информационное обслуживание и поддержку деятельности:

- заказчиков (владельцев) изделия;
- разработчиков (конструкторов), технологов, управленческого и производственного персонала предприятия-изготовителя;
- эксплуатационного и ремонтного персонала заказчика и специализированных служб.

Более подробно состав ИО, входящих в ОБДИ, представлен на рис. 17.3, согласно которому в составе ОБДИ можно условно выделить три раздела:

- нормативно-справочный;
- долговременный;
- актуальный.

Содержание *нормативно-справочного раздела* ОБДИ обновляется по мере поступления новых и отмены действующих нормативных документов.

В *долговременном разделе* должны храниться ИО, содержащие данные, аккумулирующие собственный опыт предприятия.

Долговременный раздел ОБДИ дополняется и обновляется по мере появления новых технических решений, признанных типовыми и пригодными для дальнейшего использования.

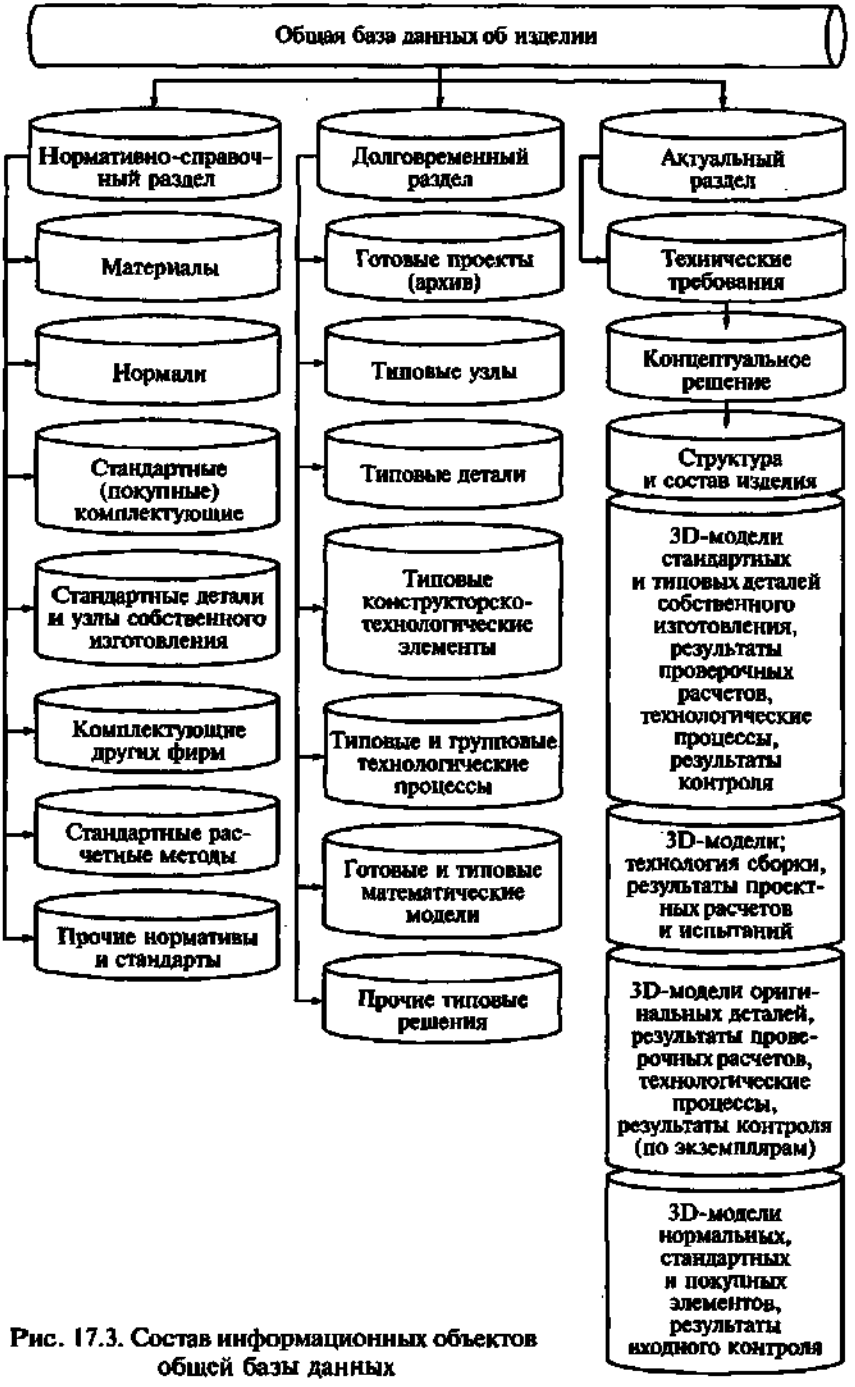


Рис. 17.3. Состав информационных объектов общей базы данных

В *актуальном разделе* (по-видимому, самом большом по объему и самом сложном по структуре) должны храниться ИО, содержащие данные об изделиях, находящихся на различных стадиях ЖЦ.

Структура этого раздела весьма приблизительна и требует развития и уточнения, в том числе разбивки на дополнительные подразделы (классификационные уровни).

Как уже отмечалось, кроме информационных объектов, относящихся (прямо или косвенно) к изделиям, в ИИС содержится информация о предприятии: о производственной и управленческой структуре, технологическом и вспомогательном оборудовании, персонале, финансах и т.д. Вся совокупность этих данных образует ОБДП, которая также состоит из нескольких разделов: экономика и финансы, внешние связи предприятия, производственно-технологическая среда предприятия, система качества.

В разделе, посвященном экономике и финансам, должны храниться информационные объекты, содержащие сведения:

- о конъюнктуре рынка изделий предприятия, включая цены и их динамику;
- состоянии финансовых ресурсов предприятия;
- ситуации на фондовом и финансовом рынках (курсы акций предприятия, биржевые индексы, процентные ставки, валютные курсы и т.д.);
- реальном и прогнозируемом портфеле заказов;
- а также прочие сведения финансово-экономического и бухгалтерского характера.

В разделе, посвященном внешним связям предприятия, должны храниться ИО, содержащие сведения о фактических и возможных поставщиках и потребителях (заказчиках). Данный раздел формируется и используется в процессе маркетинговых исследований.

В разделе, посвященном производственно-технологической среде предприятия, должны храниться информационные объекты, содержащие сведения:

- о производственной структуре предприятия;
- технологическом, вспомогательном и контрольно-измерительном оборудовании;
- транспортно-складской системе предприятия;
- энерговооруженности предприятия;
- кадрах;
- а также прочие данные о предприятии.

В разделе, посвященном системе качества, должны храниться сведения:

- о структуре действующей на предприятии системы качества;
- действующих на предприятии стандартах по качеству;
- международных и российских стандартах по качеству;
- должностных инструкциях в области качества;
- а также прочая информация о системе качества.

Из ИИС могут быть извлечены разнообразные документы, необходимые для функционирования предприятия. Документы могут быть представлены как в электронном, так и в традиционном, бумажном виде.

При электронном проектировании средствами систем CAE/CAD/CAM должны использоваться и электронные средства управления конфигурацией, отвечающие, в частности, требованиям стандарта ИСО 10303-203.

17.3. Управление интегрированной информационной средой предприятия

Управление интегрированной информационной средой предприятия предполагает, что все процессы, протекающие в ИИС, управляемы, т.е. поддаются воздействиям со стороны уполномоченных лиц (администраторов) и соответствующих программных средств. Совокупность таких средств принято называть системой управления базами данных. В функции СУБД входит:

- помещение информации в базу данных;
- хранение информации, в том числе создание резервных копий;
- обновление данных (ввод новых данных взамен данных, утративших актуальность);
- обеспечение достоверности и целостности данных;
- поиск данных по различным признакам;
- создание отчетов;
- установление (изменение) и оперативная проверка прав доступа пользователей к данным и т.д.

Распределенный характер ИИС, в отличие от традиционных БД, требует создания специальной инфраструктуры, обеспечивающей накопление, хранение и передачу данных между всеми заинтересованными участниками ЖЦ изделия. Такая инфраструктура должна представлять собой комплекс программных и аппаратных средств, позволяющий решать перечисленные выше задачи.

В рамках традиционного предприятия, расположенного на единой (и единственной) производственной площадке, такая инфраструктура создается на основе локальной вычислительной сети и соответствующего системного и прикладного программного обеспечения.

Для предприятий с географически распределенной производственной структурой, особенно для виртуальных предприятий, эта проблема очень важна.

Анализ сегодняшнего состояния телекоммуникационных средств и систем позволяет высказать утверждение, что основой инфраструктуры виртуального предприятия, а также предприятия с географически распределенной структурой может служить глобаль-

ная сеть Интернет, в которой данные передаются с помощью протокола TCP/IP. Несмотря на внешнюю простоту и доступность сети Интернет использование ее в качестве структурообразующего средства связано с рядом специфических проблем.

Первая проблема заключается в том, что для эффективного накопления, хранения и использования данных всеми участниками информационного обмена в соответствии с технологиями ИПИ хранилище данных должно быть логически локализовано в форме, которую в Интернет-технологиях принято называть порталом. Иными словами, должен быть создан специальный узел сети Интернет, предназначенный для информационного обслуживания предприятия, виртуального предприятия или корпорации.

Вторая проблема связана с тем, что этот узел и соответственно участники информационного обмена должны быть ограждены от вмешательства в этот обмен посторонних лиц и организаций даже при отсутствии у них какого-либо злого умысла или враждебных интересов.

Третья проблема заключается в защите информации от несанкционированного доступа лиц и организаций, имеющих своей целью использование этой информации во враждебных целях (в целях похищения сведений, составляющих государственную и (или) коммерческую тайну, в целях нарушения целостности и (или) достоверности данных, передаваемых участниками информационного обмена и т.д.).

17.4. Управление качеством

Система управления качеством продукции является элементом управленческой деятельности предприятия. В соответствии с международным стандартом ИСО 9000:2000 управление качеством (УКч) должно базироваться на информационной системе, поддерживающей автоматизированную обработку данных и документирование процессов обеспечения качества на всех стадиях ЖЦ изделия, автоматизированное управление этими процессами, данными и документацией. В этом смысле управление качеством становится неотъемлемой частью АСУ предприятием и может быть отнесена к CALS-технологиям. Это означает, что информация, циркулирующая в системе управления качеством, должна быть представлена в форматах, регламентированных стандартами CALS-технологий, и состоять из набора информационных объектов, входящих в ИИС предприятия.

Укрупненная структура системы управления качеством приведена на рис. 17.4, на котором показаны связи системы с объектом управления (процессами ЖЦ продукции), а также с внешней по отношению к рассматриваемой системе средой, которую в данном случае представляет некий обобщенный потребитель, чьи тре-

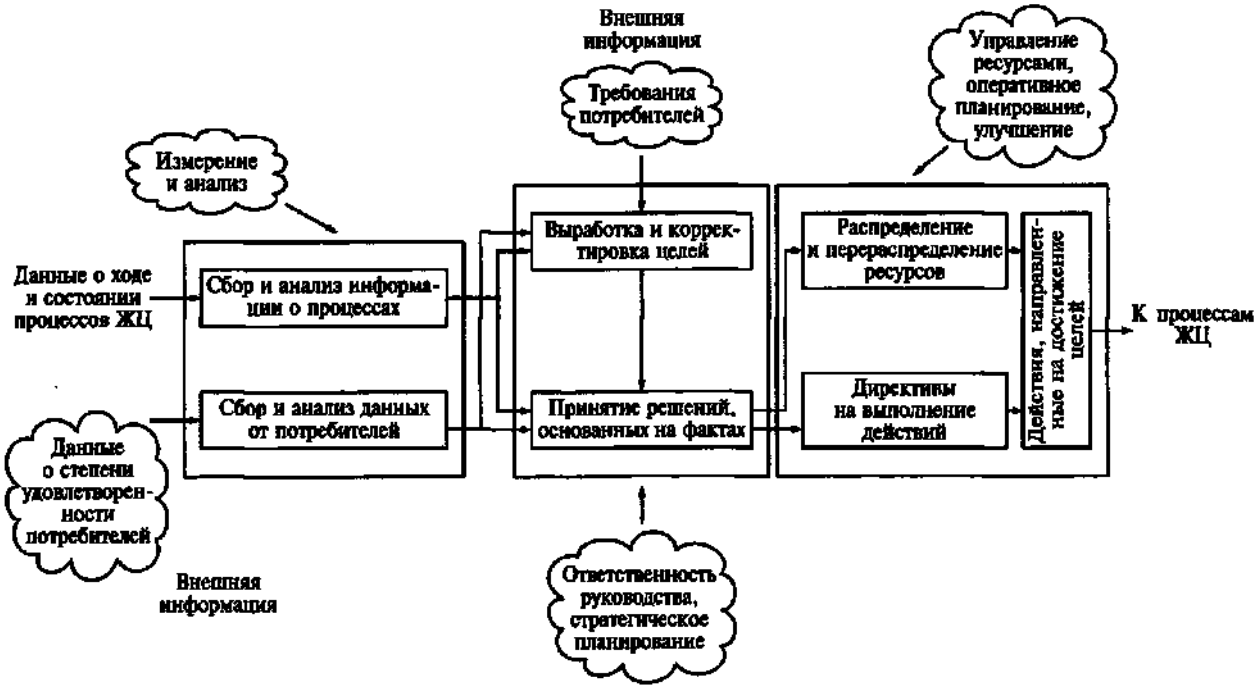


Рис. 17.4. Структура системы управления качеством

бования и степень удовлетворенности являются внешними данными.

Присутствующие в структуре блоки выработки и корректировки целей и принятия решений, основанных на фактах, вместе эквивалентны тому, что в стандарте ИСО 9000:2000 называется ответственностью руководства и планированием (в данном контексте — стратегическим). Блоки сбора и анализа данных от потребителей и информации о процессах отражают процессы, именуемые в стандарте как измерение и анализ. Группа блоков, связанных с реализацией решений (распределение и перераспределение ресурсов, директивы на выполнение действий и сами действия, направленные на достижение целей), отражает все то, что в стандарте называют управлением ресурсами, планированием (в этом контексте — оперативным) и улучшением.

Как уже отмечалось при создании и технологической подготовке производства нового изделия средствами конструкторских и технологических САПР (CAD/CAM), в ИИС создаются ИО, описывающие структуру изделия, его состав и все входящие компоненты: детали, узлы, агрегаты, комплектующие, материалы и т.д. Каждый ИО обладает набором характеристик (атрибутов), описывающих свойства реального объекта, отображением которого он является. С точки зрения УКч такими характеристиками являются технические требования и технические условия, которым должен удовлетворять реальный объект. Кроме информации об изделии в ИИС содержится информация о производственной среде предприятия, в составе которой находятся данные, относящиеся к УКч.

17.5. Управление потоками работ

Понятие «поток работ» («workflow») — одно из базовых в современной практике управления вообще и в области ИПИ — в частности. Это понятие объединяет подходы к формализации и управлению бизнес-процессами предприятия, а также программные средства, реализующие эти подходы. Популярность и многообразие реализаций технологий workflow привели к созданию в середине 1990-х гг. международной ассоциации WfMC (Workflow Management Coalition) (WfMC), занимающейся стандартами, регламентирующими требования к системам workflow и средствам их реализации.

Согласно глоссарию WfMC, бизнес-процесс — это одна или более связанных между собой процедур или операций (функций), которые совместно реализуют некую бизнес-задачу или политическую цель предприятия, как правило, в рамках организационной структуры, описывающей функциональные роли и отношения. Бизнес-процесс может целиком осуществляться в пределах одного организационного подразделения, охватывать несколько

подразделений в рамках организации или даже несколько различных организаций, как, например, в системе отношений клиент — поставщик. Бизнес-процесс может включать в себя формальные и неформальные взаимодействия между участниками; его продолжительность может изменяться в широких пределах.

Поток работ — упорядоченное во времени множество рабочих заданий, получаемых сотрудниками, которые обрабатывают эти задания вручную или с помощью средств механизации (автоматизации) в последовательности и в рамках правил, определенных для данного бизнес-процесса. Бизнес-процесс — это своего рода конвейер, работающий по своим правилам и технологиям, а поток работ (заданий) аналогичен потоку изделий (узлов, деталей), которые этот конвейер передвигает.

Бизнес-процесс объединяет поток заданий и функции, которые должны выполняться над элементами (заданиями) этого потока, людей и оборудование, которые реализуют эти функции, а также правила, управляющие последовательностью выполнения этих функций. Технология workflow призвана все это автоматизировать.

Приведем два определения из глоссария WfMC.

1. *Поток работ* — полная или частичная автоматизация бизнес-процесса, при которой документы, информация или задания передаются для выполнения необходимых действий от одного участника к другому в соответствии с набором процедурных правил.

2. *Система управления потоком работ* — система, которая описывает этот поток (бизнес-процесс), создает его и управляет им при помощи программного обеспечения, которое способно интерпретировать описание процесса, взаимодействовать с его участниками и при необходимости вызывать соответствующие программные приложения и инструментальные средства.

Такая система автоматизирует процесс, а не функцию. Появление систем управления потоком работ и соответствующих программных средств — это реакция рынка информационных технологий на внедрение новых принципов управления предприятиями. Сегодня эти принципы эволюционируют от функциональной ориентации (придуманной Адамом Смитом еще в 1776 г. и успешно работающей на протяжении более двух столетий) в направлении процессной ориентации.

Практически все предыдущие решения (чаще всего реализованные в технологиях локальных СУБД) позволяли достаточно эффективно автоматизировать отдельные операции и функции, а не процесс. В рамках этих решений сотрудники, сидя за своими компьютерами (или терминалами), обмениваются информацией с базами данных и между собой, получают данные, справки, документы, формируют отчеты. При этом последовательность действий сотрудников и правила их взаимодействия определены в

лучшем случае инструкциями, а за правильностью и сроками их выполнения следит вышестоящее начальство. Информационная система все это никак не поддерживает.

Процессный подход заставил руководство предприятий сконцентрировать внимание на правилах взаимодействия участников процесса, так как именно взаимодействия в силу своей размытости и недостаточной определенности являются основным источником неоправданных издержек и потерь. Потребность в средствах автоматического отслеживания порядка и времени выполнения отдельных функций (операций), маршрутов документов, занятости сотрудников на различных стадиях процесса привели к созданию разнообразных систем управления потоками работ и к утверждению этого понятия в качестве одного из базовых инвариантов концепции ИПИ.

Контрольные вопросы

1. Что означает термин «интегрированная информационная среда»?
2. Что означает термин «информационный объект»?
3. Какая информация должна содержаться в общей базе данных об изделии?
4. Какая информация должна содержаться в общей базе данных предприятия?
5. Какие задачи и в соответствии с каким стандартом решает Система управления качеством?
6. Какая связь существует между понятиями «управление потоками работ» и «бизнес-процессы»?

Глава 18

БАЗЫ ДАННЫХ В СИСТЕМАХ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

18.1. Базы данных в конструкторских системах автоматизированного проектирования

Современные системы конструкторского проектирования, такие как отечественная T-FLEX CAD или американская Solid Works, представляют собой системы параметрического черчения. Сущность данного метода проектирования чертежей нового изделия построена на создании конструкторских баз данных, содержащих чертежи типовых изделий, все параметры которых могут быть выражены с помощью переменных или рассчитаны с помощью формул. Эти параметры заносятся в базу данных (таблицы).

Проектирование конструкции нового изделия в этом случае осуществляется в следующей последовательности.

1. Из конструкторской базы данных выбирается изделие — аналог.
2. В таблицу параметрической базы данных вводятся параметры нового изделия.
3. Система на основе введенных параметров формирует новый чертеж, сохраняя его в конструкторской базе, а его параметры — в параметрической базе данных.

Процесс создания параметрической базы данных можно разделить на следующие действия:

1) формирование структуры таблицы базы данных;
2) создание переменных, в том числе на основе баз данных;
3) создание интерфейса пользователя при выборе параметров для формирования чертежа и ввода новых данных.

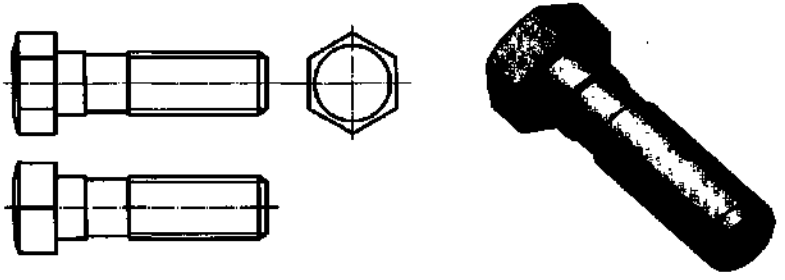


Рис. 18.1. Болт по ГОСТ 7795—70

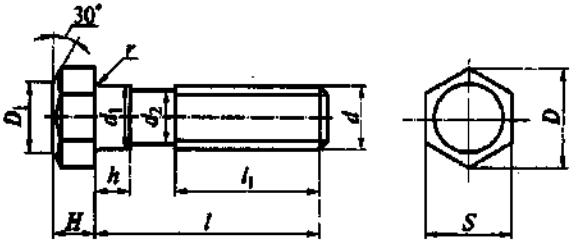


Рис. 18.2. Параметры, определяющие размеры болта

Создание базы данных рассмотрим на следующем примере. Пусть это будет болт по ГОСТ 7795—70 (рис. 18.1).

На рис. 18.2, 18.3 показаны примеры формирования параметрической базы данных для проектирования детали типа болт.

Параметры этого болта в зависимости от введенного диаметра и длины должны выбираться из базы данных. Болт имеет несколько исполнений, в соответствии с которыми должны меняться чертежи. Болт будет использоваться в качестве фрагментов в других чертежах.

При создании базы данных необходимо создать так называемые внутренние базы данных, из которых будут выбираться значения. Сначала необходимо решить, в зависимости от какого параметра будут выбираться остальные. На рис. 18.2 показаны параметры, определяющие размеры болта.

В нашем случае один из таких параметров — диаметр резьбы болта. Итак, первый столбец — диаметр болта, а потом идут все остальные (кроме длин). Имена столбцов базы данных (и самой базы данных) необходимо выбирать так, чтобы потом, при создании переменных, вы могли легко вспомнить, за какой параметр отвечает каждый столбец, но лучше не делать имена столбцов слишком длинными. Логичнее назвать столбцы так, как наз-

	d	d1	h	S	H	D	r	l
1	6	6	3	10	4,20	10,90	0,25	2,00
2	8	8	4	12	5,60	13,10	0,40	2,50
3	10	10	5	14	7,00	15,30	0,40	3,50
4	12	12	6	17	8,00	18,70	0,60	4,00
5	16	16	8	22	10,00	24,30	0,80	5,00
6	20	20	10	27	13,00	29,90	0,80	6,50
7	24	24	12	32	15,00	35,00	0,80	7,50
8	30	30	16	41	19,00	45,20	1,00	9,50
9	36	36	18	50	23,00	55,40	1,00	11,50
10	42	42	21	60	26,00	66,40	1,20	13,00
11	48	48	24	70	30,00	77,70	1,60	15,00

Рис. 18.3. Фрагмент таблицы параметров болта

FROM [Переход (Пайка)]
WHERE ((([Переход (Пайка)].[Код пер]) = 0 Or ([Переход (Пайка)].[Код пер]) = 1 Or ([Переход (Пайка)].[Код пер]) = 4 Or ([Переход (Пайка)].[Код пер]) = 13 Or ([Переход (Пайка)].[Код пер]) = = 16 Or ([Переход (Пайка)].[Код пер])=19 Or ([Переход (Пайка)].[Код пер]) = 39 Or ([Переход (Пайка)].[Код пер]) = 41 Or ([Переход (Пайка)].[Код пер]) = 46 Or ([Переход (Пайка)].[Код пер]) = 48))
ORDER BY [Переход (Пайка)].[Код пер];

Отметим условия отбора данных из таблицы переходов, связанных условием Or:

[Переход (Пайка)].[Код пер])=0 Or ([Переход (Пайка)].[Код пер]) = 1 ...

Применение условия отбора с логическим оператором OR, позволяет расставить содержание записей в последовательности, отвечающей заданному набору условий.

В результате выполнения запроса откроется динамическая таблица (рис. 18.10), содержащая только состав и последовательность переходов для выбранного варианта ТП.

На рис. 18.11, 18.12 показан фрагмент определения причин дефектов паяного соединения.

18.3. Экспертные компьютерные системы

Основной задачей любого предприятия является снижение затрат на технологическую подготовку и управление производством. Одна из серьезных проблем состоит в быстрой оценке предполагаемых трудовых и материальных затрат на изготовление новых изделий. Решить эту проблему можно двумя путями:

1) привлечением высококвалифицированных специалистов;

2) созданием компьютерных экспертных систем на основе технологических баз данных.

В настоящее время многие предприятия сталкиваются с проблемой нехватки квалифицированных специалистов в области управленческой деятельности, поэтому разработка экспертных систем в области принятия управленческих решений является одной из приоритетных задач.

Для разработки экспертных систем эффективно использовать системы управления реляционными базами данных, например Microsoft Access.

Рассмотрим экспертную систему для оценки ожидаемых затрат на изготовление деталей новых изделий.

Предлагаемая экспертная система строится на следующих исходных положениях.

1. На предприятии все детали основного производства имеют технологический код согласно Технологического классификатора деталей приборостроения и машиностроения (ТКД).

2. На предприятии имеется технологическая база данных, содержащая по каждой детали маршрутный технологический процесс, в составе которого имеются данные о трудоемкости изготовления детали.

Исходя из этих положений прогнозирование ожидаемой трудоемкости новых деталей сводится к следующим действиям.

1. Присвоить технологический код детали.

2. Найти в базе данных аналоги — детали с аналогичным технологическим кодом.

3. Рассчитать среднюю трудоемкость всех деталей данного технологического кода, имеющихся в базе данных.

4. Присвоить новой детали рассчитанную среднюю трудоемкость.

Как показали результаты, погрешность экспертных оценок на основе технологической классификации составляет не более 20 %, что вполне удовлетворительно для принятия решений о целесообразности изготовления нового изделия в условиях конкретного предприятия.

Одной из задач при разработке экспертной системы является разработка системы автоматизированного кодирования деталей в соответствии с Технологическим классификатором.

Технологический классификатор деталей машиностроения и приборостроения был создан еще в 1981 — 1985 гг. в соответствии с Программой работ по ЕСКК и ЕСУД, утвержденной постановлением ГКНТ и Госпланом СССР.

Широкое использование технологического классификатора деталей в промышленности при подготовке производства в рамках внедрения ЕСТПП показало его высокую эффективность при решении производственных задач с применением средств вычислительной техники и новейших технико-математических методов.

Технологический классификатор деталей используется в системе подготовки производства и управления им совместно с общесоюзными классификаторами технико-экономической информации. Опыт его внедрения в отраслях промышленности показал, что он создает предпосылки для решения следующих основных задач:

- анализ номенклатуры деталей по их конструкторско-технологическим характеристикам;
- группирование деталей по конструкторско-технологическому подобию для разработки типовых и групповых технологических процессов с использованием ЭВМ;
- подетальная специализация производственных подразделений (участков, цехов, заводов);
- повышение серийности и концентрация производства деталей;

• унификация и стандартизация деталей и технологических процессов их изготовления;

• рациональный выбор типов технологического оборудования;

• тематический поиск и заимствование ранее разработанных типовых или групповых технологических процессов;

• автоматизация проектирования деталей и технологических процессов их изготовления.

Основной целью ТКД является снижение трудоемкости и сокращение сроков технологической подготовки производства.

Технологический классификатор, используемый в настоящее время в промышленности совместно с классами 40 и 50 «Детали общемашиностроительного применения», не отвечает полностью современным условиям производства деталей, так как распространяется на достаточно ограниченный круг деталей, в основном общемашиностроительного характера.

Настоящий Технологический классификатор деталей машиностроения и приборостроения при неизменных основных принципах его построения охватывает детали всех отраслей промышленности основного и вспомогательного производств.

Он является логическим продолжением и дополнением классов деталей Классификатора ЕСКД (классы 71, 72, 73, 74, 75, 76).

Классификатор ЕСКД разработан в качестве информационной части ГОСТ 2.201—80 «ЕСКД. Обозначение изделий и конструкторских документов» единой классификационной обезличенной системы обозначения изделий и конструкторских документов машиностроения и приборостроения. Классы деталей создают оптимальные условия:

• для создания единого информационного языка для автоматизированных систем управления и облегчения тематического поиска деталей и их конструкторских документов с целью предотвращения разработки аналогичных;

• определения объектов и направлений унификации и стандартизации;

• обеспечения возможности использования различными предприятиями и организациями конструкторской документации в проектировании, производстве, эксплуатации, ремонте, разработанной другими организациями, без ее переоформления;

• широкого внедрения средств вычислительной техники в сферы проектирования и управления;

• применения кодов деталей по классам деталей совместно с технологическими кодами при решении задач технологической подготовки производства с использованием средств вычислительной техники.

Применение данного классификатора должно быть одним из основных методов создания и внедрения на предприятиях CALS-технологий.

Технологический классификатор устанавливает технологические коды деталей:

- изготовляемых литьем;
- изготовляемых ковкой и объемной штамповкой;
- изготовляемых листовой штамповкой;
- обрабатываемых резанием;
- термически обрабатываемых;
- изготовляемых формообразованием из полимерных материалов и резины;
- с покрытием;
- обрабатываемых электрофизико-химическими методами;
- изготовляемых порошковой металлургией.

В основу технологической классификации деталей положен фасетный метод, при котором заданное множество делят на группировки по различным признакам классификации.

Технологическую классификацию распространяют на детали машиностроения и приборостроения основного и вспомогательного производств.

В качестве классификационных признаков используют существенные технологические характеристики деталей, которые в сочетании с конструктивными признаками определяют их технологическое подобие. Классификационные таблицы-фасеты составляют для основных признаков технологической классификации и признаков, характеризующих вид детали по технологическому методу ее изготовления.

Детали кодируют буквенно-цифровым кодом. В структуре технологического кода деталей за каждым признаком закрепляют определенный разряд (позицию) и число знаков. Система построения кодовых обозначений обеспечивает формирование групп, состоящих из оптимального числа деталей, с использованием средств вычислительной техники.

Структура конструкторско-технологического кода обеспечивает обработку информации в различных кодовых комбинациях для решения производственных задач и допускает использование частей кода и их сочетаний в зависимости от характера решаемых задач.

Технологический классификатор представляет собой систематизированный перечень наименований общих признаков деталей, их составляющих частных признаков и кодовых обозначений в виде классификационных таблиц. Форма классификатора допускает оперативное изменение его содержания без изменения других позиций классификатора и его переиздания.

Технологическое кодовое обозначение детали включает в себя 14 знаков. Это кодовое обозначение состоит из двух частей: кодового обозначения классификационных группировок основных признаков (постоянная часть) — шесть знаков; кодового обозначения классификационных группировок признаков, характеризу-

Таким образом, данная система позволяет также осуществлять интеллектуальный анализ данных, поскольку имеется возможность оценить разброс значений трудоемкости деталей, отнесенных к одной группе. Это решает две важные задачи:

- позволяет прогнозировать погрешности оценки трудоемкости новой детали;
- позволяет выявить ошибки в классификации деталей, а также ошибки в определении трудоемкости детали-аналога.

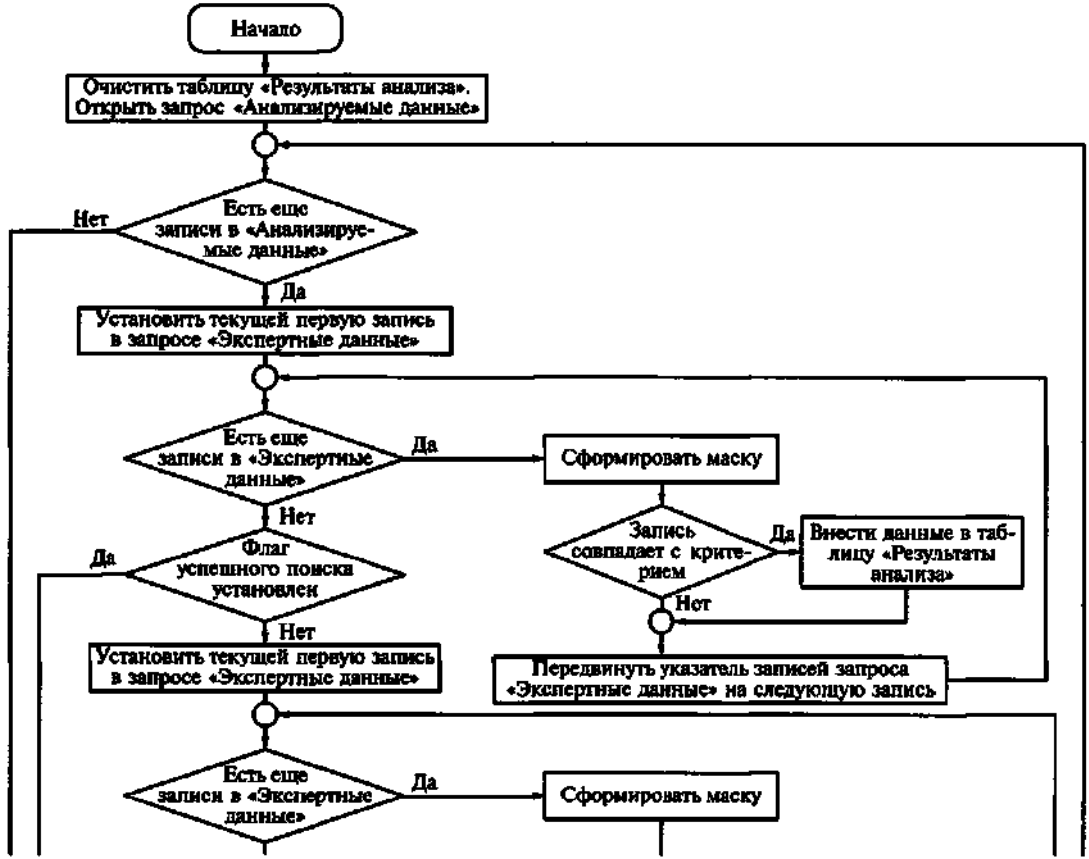
Блок-схема алгоритма экспертной оценки трудоемкости изготовления деталей приведена на рис. 18.22.

Приведем текст программы на VBA для анализа данных в сформированной таблице и экспертной оценки трудоемкости на новые детали с необходимыми комментариями:

```
Option Compare Database
Option Explicit

Private Sub Кнопка0_Click()
On Error GoTo Err_Кнопка1_Click
Dim dbs As Database, Rst1 As Recordset, Rst2 As Recordset, Rst3 As Recordset, Fj As Boolean, stDocName As String
stDocName = "Очистка результатов анализа"
DoCmd.OpenQuery stDocName, acNormal, acEdit ' Очистка таблицы "Результаты анализа"'
Set dbs = CurrentDb ' Возвращает объектную переменную типа Database, представляющую текущую базу данных.'
Set Rst1 = dbs.OpenRecordset("Анализируемые данные") ' Создает новый объект Recordset для доступа к данным таблицы "Анализируемые данные" с именем RST1 и добавляет его в семейство Recordsets.'
Set Rst2 = dbs.OpenRecordset("Результаты анализа") ' Создает новый объект Recordset для доступа к данным таблицы "Результаты анализа" с именем RST2 и добавляет его в семейство Recordsets.'
Set Rst3 = dbs.OpenRecordset("Экспертные данные") ' Создает новый объект Recordset для доступа к данным таблицы "Экспертные данные" с именем RST3 и добавляет его в семейство Recordsets.'
' Начало внешнего цикла. В цикле последовательно выбираются записи таблицы "Анализируемые данные"
' и сравниваются с данными таблицы "Экспертные данные".
Do Until Rst1.EOF
Fj = False 'Переменная Fj указывает на результаты поиска'
Rst3.MoveFirst 'Перед каждым циклом поиска указатель записей таблицы "Экспертные данные" устанавливается на первую запись'
Do Until Rst3.EOF 'Внутренний цикл. В цикле последовательно выбираются записи таблицы "Экспертные данные"
If (Rst1!Констр Like Rst3!Констр) And (Rst1!ПостКод Like Rst3!ПостКод) And (Rst1!СформКод Like Rst3!СформКод) Then
```

```
Fj = True 'Если записи в таблицах совпали, то устанавливается признак успешного поиска.'
Rst2.AddNew 'В таблицу "Результаты анализа добавляется новая запись"'
Rst2!Анализир = (Rst1!Констр) & "." & (Rst1!ПостКод) & "." & (Rst1!СформКод) 'Содержащая информацию о технологическом коде анализируемой детали,'
Rst2!Эксперт = (Rst3!Констр) & "." & (Rst3!ПостКод) & "." & (Rst3!СформКод) 'Коде аналога, а также номера чертежей анализируемой детали и аналога'
Rst2!НомерЧертежаАн = Rst1!НомерЧертежа
Rst2!НомерЧертежаЭксп = Rst3!НомерЧертежа
Rst2!Процент = "100 %"
Rst2!Трудоемкость = Rst3!Трудоемкость 'и трудоемкость изготовления детали-аналога.'
Rst2.Update
End If
Rst3.MoveNext
Loop
If Fj = False Then 'Если на предыдущем цикле поиска аналог детали найден не был, то осуществляется повторный поиск. При этом некоторые позиции технологического кода не учитываются'
Rst3.MoveFirst 'Внутренний цикл. В цикле последовательно выбираются записи таблицы "Экспертные данные"'
Do Until Rst3.EOF 'Маска поиска формируется в функции GetStringJ. При этом учитывается технологический метод изготовления детали'
If (Rst1!Констр = Rst3!Констр) And (((Rst1!ПостКод) & (Rst1! СформКод)) Like GetStringJ((Rst3!ПостКод) & (Rst3!СформКод))) Then
Fj = True 'Если записи в таблицах совпали, то устанавливается признак успешного поиска.'
Rst2.AddNew 'В таблицу "Результаты анализа добавляется новая запись"'
Rst2!Анализир = (Rst1!Констр) & "." & (Rst1!ПостКод) & "." & (Rst1!СформКод) 'Содержащая информацию о технологическом коде анализируемой детали,'
Rst2!Эксперт = (Rst3!Констр) & "." & (Rst3!ПостКод) & "." & (Rst3!СформКод) 'Коде аналога, а также номера чертежей анализируемой детали и аналога'
Rst2!Процент = "80 %"
Rst2!НомерЧертежаАн = Rst1!НомерЧертежа
Rst2!НомерЧертежаЭксп = Rst3!НомерЧертежа
Rst2!Трудоемкость = Rst3!Трудоемкость 'и трудоемкость изготовления детали-аналога.'
Rst2.Update
End If
```


Начало
Очистить таблицу «Результаты анализа». Открыть запрос «Анализируемые данные»
Есть еще записи в «Анализируемые данные»
Нет
Да
Установить текущей первую запись в запросе «Экспертные данные»
Есть еще записи в «Экспертные данные»
Да
Сформировать маску
Нет
Запись совпадает с критерием
Да
Внести данные в таблицу «Результаты анализа»
Нет
Флаг успешного поиска установлен
Да
Нет
Установить текущей первую запись в запросе «Экспертные данные»
Передвинуть указатель записей запроса «Экспертные данные» на следующую запись
Есть еще записи в «Экспертные данные»
Да
Сформировать маску

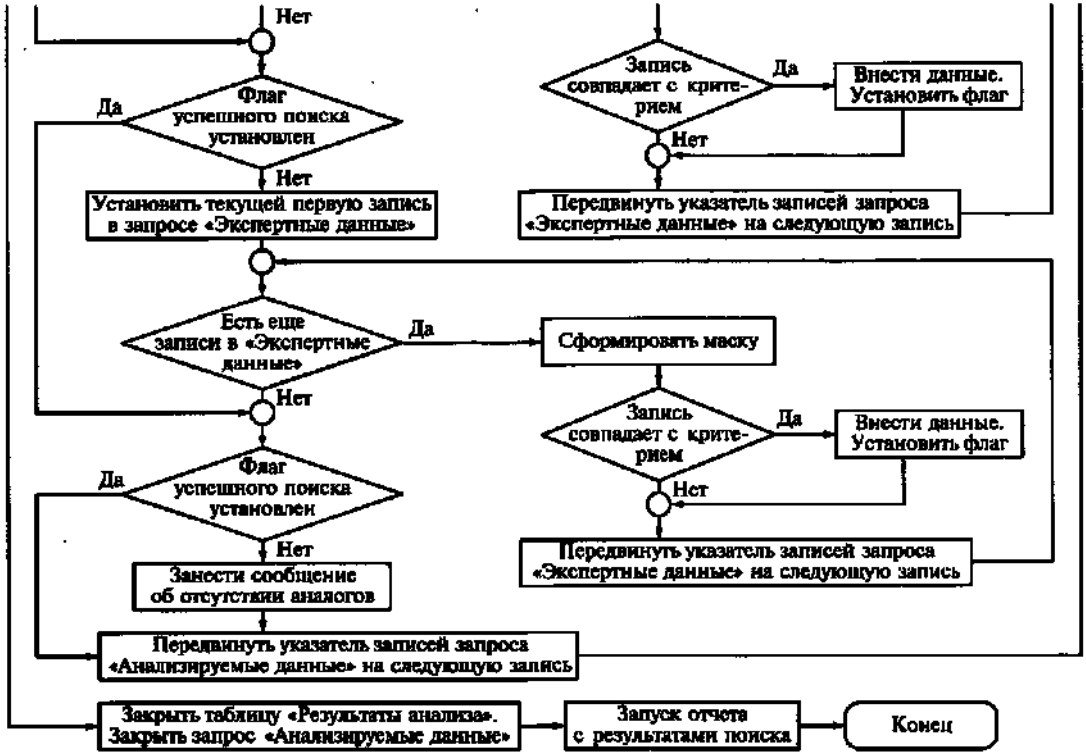


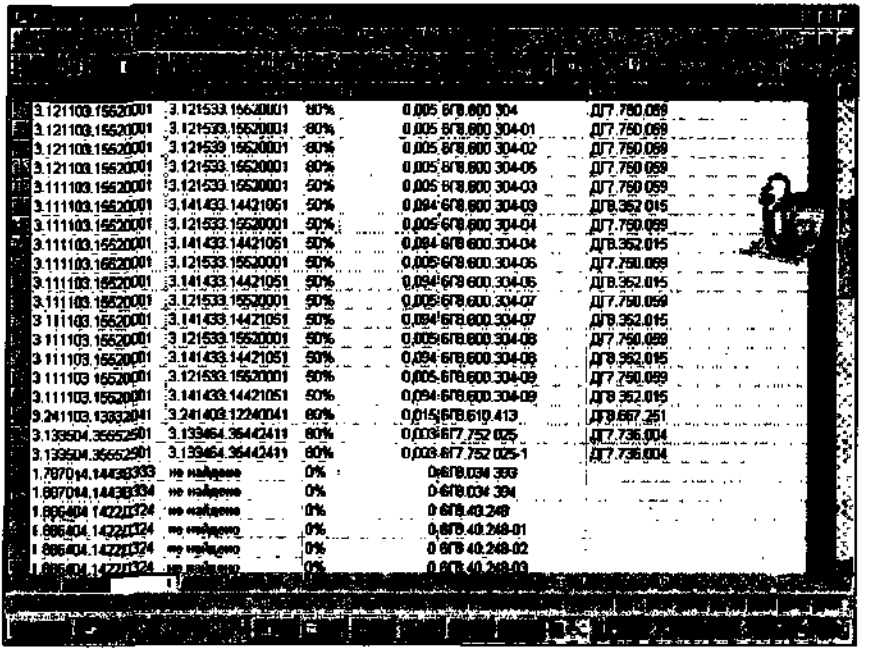
Рис. 18.22. Блок схема алгоритма экспертной оценки трудоемкости изготовления деталей

```
    Rst3.MoveNext
    Loop
    End If
    If Fj = False Then
    Rst3.MoveFirst
    Do Until Rst3.EOF
    If (Rst1!Констр = Rst3!Констр) And (((Rst1!ПостКод) & (Rst1!
Сформ-Код)) Like GetStringS((Rst3!ПостКод) & (Rst3! Сформ-
Код))) Then
    Fj = True
    Rst2.AddNew
    Rst2!Анализир = (Rst1!Констр) & "." & (Rst1!ПостКод) & "." &
(Rst1!СформКод)
    Rst2!Эксперт = (Rst3!Констр) & "." & (Rst3!ПостКод) & "." &
(Rst3!СформКод)
    Rst2!Процент = "60 %"
    Rst2!НомерЧертежаАн = Rst1!НомерЧертежа
    Rst2!НомерЧертежаЭксп = Rst3!НомерЧертежа
    Rst2!Трудоемкость = Rst3!Трудоемкость
    Rst2.Update
    End If
    Rst3.MoveNext
    Loop
    End If
    If Fj = False Then
    Rst3.MoveFirst
    Do Until Rst3.EOF
    If (Rst1!Констр = Rst3!Констр) And (((Rst1!ПостКод) & (Rst1!
СформКод)) Like GetStringK((Rst3!ПостКод) & (Rst3! Сформ-
Код))) Then
    Fj = True
    Rst2.AddNew
    Rst2!Анализир = (Rst1!Констр) & "." & (Rst1!ПостКод) & "." &
(Rst1!СформКод)
    Rst2!Эксперт = (Rst3!Констр) & "." & (Rst3!ПостКод) & "." &
(Rst3!СформКод)
    Rst2!Процент = "50 %"
    Rst2!НомерЧертежаАн = Rst1!НомерЧертежа
    Rst2!НомерЧертежаЭксп = Rst3!НомерЧертежа
    Rst2!Трудоемкость = Rst3!Трудоемкость
    Rst2.Update
    End If
    Rst3.MoveNext
    Loop
    End If
```

```
If Fj = False Then 'Если аналог для детали не найден, то в таблицу "Результаты анализа"'
Rst2.AddNew 'Заносится информация об отсутствии аналога для данной детали'
Rst2!Анализир = (Rst1!Констр) & "." & (Rst1!ПостКод) & "." & (Rst1!СформКод)
Rst2!Эксперт = "не найдено"
Rst2!НомерЧертежаАн = Rst1!НомерЧертежа
Rst2!Процент = "0 %"
Rst2.Update
End If
Rst1.MoveNext
Loop
Rst1.Close 'Закрывает открытый объект доступа к данным таблицы "Анализируемые данные".'
Rst2.Close 'Закрывает открытый объект доступа к данным таблицы "Результаты анализа".'
Rst3.Close 'Закрывает открытый объект доступа к данным таблицы "Экспертные данные".'
stDocName = "Результаты анализа"
DoCmd.OpenReport stDocName, acPreview 'Формирует отчет на основе таблицы "Результаты анализа"'
Exit_Кнопка1_Click:
Exit Sub
Err_Кнопка1_Click:
MsgBox Err.Description
Resume Exit_Кнопка1_Click
End Sub
Public Function GetStringJ(Expert As String) As String
GetStringJ = Mid(Expert, 1, 3) & "??" & Mid(Expert, 6, 2) & "???????"
End Function
Public Function GetStringS(Expert As String) As String
GetStringS = Mid(Expert, 1, 2) & "???" & Mid(Expert, 6, 2) & "???????"
End Function
Public Function GetStringK(Expert As String) As String
GetStringK = Mid(Expert, 1, 1) & "?" & Mid(Expert, 1, 1) & "??" & Mid(Expert, 6, 2) & "???????"
End Function
```

На рис. 18.23 показан фрагмент таблицы результатов экспертной оценки трудоемкости изготовления новых деталей. Из данной таблицы следует, что ряд деталей имеет прогнозируемую оценку трудоемкостью с точностью 80 %, ряд деталей — с точностью 50 %, а для некоторых деталей аналоги не найдены.

Данная система была апробирована на Втором Московском приборостроительном заводе. Был поставлен ряд экспериментов, во



3.121109.15520001	3.121533.15520001	80%	0.005.678.600.304		ДТ7.750.059
3.121109.15520001	3.121533.15520001	80%	0.005.678.600.304-01		ДТ7.750.059
3.121109.15520001	3.121533.15520001	80%	0.005.678.600.304-02		ДТ7.750.059
3.121109.15520001	3.121533.15520001	80%	0.005.678.600.304-05		ДТ7.750.059
3.111103.15520001	3.121533.15520001	50%	0.005.678.600.304-03		ДТ7.750.059
3.111103.15520001	3.141433.14421051	50%	0.094.678.600.304-09		ДТ8.352.015
3.111103.15520001	3.121533.15520001	50%	0.005.678.600.304-04		ДТ7.750.059
3.111103.15520001	3.141433.14421051	50%	0.094.678.600.304-04		ДТ8.352.015
3.111103.15520001	3.121533.15520001	50%	0.005.678.600.304-06		ДТ7.750.059
3.111103.15520001	3.141433.14421051	50%	0.094.678.600.304-06		ДТ8.352.015
3.111103.15520001	3.121533.15520001	50%	0.005.678.600.304-07		ДТ7.750.059
3.111103.15520001	3.141433.14421051	50%	0.094.678.600.304-07		ДТ8.352.015
3.111103.15520001	3.121533.15520001	50%	0.005.678.600.304-08		ДТ7.750.059
3.111103.15520001	3.141433.14421051	50%	0.094.678.600.304-08		ДТ8.352.015
3.111103.15520001	3.121533.15520001	50%	0.005.678.600.304-09		ДТ7.750.059
3.111103.15520001	3.141433.14421051	50%	0.094.678.600.304-09		ДТ8.352.015
9.241103.13332041	3.241409.12240041	80%	0.015.678.610.413		ДТ9.667.251
3.139504.35652501	3.139464.36442411	80%	0.003.677.752.025		ДТ7.736.004
3.139504.35652501	3.139464.36442411	80%	0.003.677.752.025-1		ДТ7.736.004
1.787044.14438333	не найдено	0%	0.678.034.359		
1.887044.14438334	не найдено	0%	0.678.034.394		
1.886404.14220324	не найдено	0%	0.678.40.248		
1.886404.14220324	не найдено	0%	0.678.40.248-01		
1.886404.14220324	не найдено	0%	0.678.40.248-02		
1.886404.14220324	не найдено	0%	0.678.40.248-03		

Рис. 18.23. Фрагмент таблицы результатов экспертной оценки трудоемкости изготовления новых деталей

время которых одно из изделий завода было принято за аналог. Каждая из его деталей была закодирована и внесена в базу данных.

Другое изделие выступало в роли оцениваемого изделия. В ходе эксперимента стояла задача оценить трудоемкость изготовления анализируемого изделия и оценить точность прогноза трудоемкости его изготовления.

Результаты показали, что разработанная экспертная система позволяет прогнозировать трудоемкость изготовления деталей с точностью не менее ±20 %.

Контрольные вопросы

1. В чем состоит сущность параметрического проектирования чертежей?
2. В чем состоит назначение баз данных в системах параметрического черчения?
4. Назовите возможные области создания экспертных систем с применением СУБД.

СПИСОК ЛИТЕРАТУРЫ

1. *Боггс У., Боггс. М.* UML и Rational Rose: Пер. с англ. — М.: Лорри, 2000.

2. *Бусленко Н. П., Калашников В. В., Коваленко И. Н.* Лекции по теории больших систем. — М.: Сов. радио, 1973.

3. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя: Пер. с англ. — М.: ДМК, 2000.

4. *Вейскас Д.* Эффективная работа с Microsoft Access 2: Пер. с англ. — СПб.: Питер, 1995.

5. *Горев А., Ахаян Р., Макашаринов С.* Эффективная работа с СУБД. — СПб.: Питер, 1997.

6. *Карпова Т. С.* Базы данных: модели, разработка, реализация. — СПб.: Питер, 2001.

7. *Кирстен В., Ирингер М.* СУБД Cache — объектно-ориентированная разработка приложений. — СПб.: Питер, 2001.

8. *Китов А. И.* Программирование информационно-логических задач. — М.: Сов. радио, 1967.

9. *Маклаков С. В.* CASE-средства разработки информационных систем. — М.: Диалог — МИФИ, 2000.

10. Тихомиров Ю. Microsoft SQL Strver 7.0. — СПб.: BNV — Санкт-Петербург, 1999.

11. Толковый словарь по вычислительным системам / Под ред. В. Иллингуорта и др.; Пер. с англ. А. К. Белоцкого и др.; Под ред. Е. К. Масловского. — М.: Машиностроение, 1990.

12. *Фуфаев Э. В., Фуфаева Л. И.* Пакеты прикладных программ. — М.: Издательский центр «Академия», 2004.

13. *Харитонова И., Михеева В.* Microsoft Access 2000. — СПб.: БХВ — Петербург, 2001.

14. *Хорафас Д., Легг С.* Конструкторские базы данных / Пер. с англ. Д. Ф. Миронова. — М.: Машиностроение, 1990.

15. *Шапошников И.* Web-сервисы Microsoft.Net. — СПб.: БХВ — Петербург, 2002.

ОГЛАВЛЕНИЕ