БАКАЛАВР. ПРИКЛАДНОЙ КУРС

Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской

БАЗЫ ДАННЫХ

УЧЕБНИК

2-е издание

1886

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» им. В. И. УЛЬЯНОВА (ЛЕНИНА)

УМО ВО рекомендует
УМО рекомендует

Юрайт издательство

biblio-online.ru

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)

Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской

БАЗЫ ДАННЫХ

УЧЕБНИК ДЛЯ ПРИКЛАДНОГО БАКАЛАВРИАТА

2–е издание

Рекомендовано Учебно–методическим отделом высшего образования в качестве учебника для студентов высших учебных заведений, обучающихся по инженерно–техническим направлениям и специальностям

Рекомендовано Учебно–методическим объединением вузов по университетскому политехническому образованию в качестве учебника для студентов вузов, обучающихся по направлениям «Информатика и вычислительная техника» и «Информационные системы»

Москва ▪ Юрайт ▪ 2015

УДК 002.52
ББК 32.81я73
С56

***Рецензент*:**

Игнатьев М. Б. — доктор технических наук, профессор Санкт-Петербургского государственного университета аэрокосмического приборостроения, лауреат Государственной премии РФ, заслуженный деятель науки и техники РФ.

Советов, Б. Я.

С56 Базы данных : учебник для прикладного бакалавриата / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 2-е изд. — М. : Издательство Юрайт, 2015. — 463 с. — Серия : Бакалавр. Прикладной курс.

ISBN 978-5-9916-4685-7

В работе изложены вопросы построения и использования технологии баз данных в процессе выработки и принятия решений.

Рассмотрены как устоявшиеся теоретические вопросы, так и новые аспекты, мало или несистемно отраженные в отечественной и переводной литературе. Это относится к локальным, распределенным и объектно-ориентированным базам данных, хранилищам данных. Подробно проанализирован режим «клиент — сервер», в том числе в удаленном варианте.

Учебник отличается системным рассмотрением теоретических вопросов, которые сопровождаются компьютерной реализацией. Это позволяет лучше понять процедуры построения, работы и использования баз данных.

Соответствует актуальным требованиям Федерального государственного образовательного стандарта высшего образования.

Для студентов вузов, обучающихся по направлениям «Информатика и вычислительная техника» и «Информационные системы». Представляет несомненный интерес для разработчиков и пользователей баз данных, преподавателей и научных сотрудников, сферой деятельности которых является технология хранения и использования данных; менеджеров и руководителей различного ранга, желающих самостоятельно ознакомиться с современным состоянием технологии баз данных.

УДК 002.52
ББК 32.81я73

ISBN 978-5-9916-4685-7

Предисловие

В силу все более широкого распространения персональных компьютеров важность организации информации в виде баз данных непрерывно возрастает.

Вместе с тем в области баз данных за последние десять-пятнадцать лет произошли серьезные изменения теоретического и прежде всего прикладного характера, не нашедшие системного отражения в учебной литературе. Сюда относятся проблемы, связанные с распределенными, объектно-ориентированными и объектно-реляционными базами данных, средствами автоматизации проектирования и программирования баз данных, языком структурированных запросов SQL, режимом клиент—сервер, хранилищами данных и с многими другими.

Период 90-х годов XX в. характеризовался появлением многочисленной прикладной литературы по применяемым СУБД, в которой акцент делался на их общее описание и преимущественно на реакцию программных продуктов на нажатие конкретных кнопок и элементов меню. Изложению общей теории практически не отводилось места. Такое положение существенно затрудняло понимание пользователями (и даже разработчиками) баз данных не только работы программных средств, но и принципов и правил их построения. Более того, в таких публикациях часто вводились новые термины, порой не определяемые или имеющие разные названия в различных источниках.

Можно понять возникающие при этом сложности, если учесть, что ряд понятий и положений в области баз данных — в силу динамичного их развития — существенно трансформировался. К тому же много отдельных вопросов по базам данных опубликованы в труднодоступных журналах, прежде всего иностранных.

Одной из причин, побудивших авторов написать данную работу, является разночтение в терминологии не только в разных моделях данных, но даже в рамках программных продуктов одной и той же фирмы-разработчика, что затрудняет изложение материала преподавателями и его понимание студентами.

Главной целью данной работы является систематизация и упорядочение имеющегося материала по базам данных. Основой работы является курс лекций, читаемый авторами на протяжении ряда последних лет, проводимый лабораторный цикл по базам данных и личный опыт авторов.

Одной из базовых дисциплин общепрофессиональной подготовки для направлений подготовки дипломированных специалистов 654600 «Информатика и вычислительная техника» и 654700 «Информационные системы» является дисциплина «Базы данных», учебник по которой и предлагается авторами вниманию читателей. С целью более глубокого изучения моделей, методов и средств информационных технологий читатель может обратиться к литературе, указанной в книге.

Авторы благодарны лауреату Государственной премии, заслуженному деятелю науки и техники РФ, д-ру техн. наук, проф. М.Б. Игнатьеву, взявшему на себя труд по рецензированию учебника и сделавшему ценные замечания, способствующие улучшению его содержания.

Замечания и пожелания по содержанию книги направлять по адресу: 127994, Москва, ГСП-4, Неглинная ул., д. 29/14, издательство «Высшая школа».

Авторы

Введение

Основные теоретические разработки по базам данных относятся к 1983—1987 гг.

Они нашли свое отражение в публикациях [7—13], среди которых выделяются такие фундаментальные работы, как [11—13].

В то же время изучение только теории БД без сопровождения процедурами создания реальных баз данных вызывает у работающих серьезные сложности прикладного характера.

Потребовались, таким образом, работы с системным изложением материала от теории до практики, что особенно актуально в силу сложности материала по базам данных. Такие заслуживающие внимания работы появились в конце 90-х годов XX в. [1—6, 17].

Работа [4] посвящена преимущественно теории централизованных реляционных баз данных, характеризуется большим количеством упражнений и опирается при проектировании БД на специфические ER-диаграммы. Из других многочисленных аспектов баз данных в работе конспективно освещены лишь особенности объектно-ориентированных языков запросов.

Только централизованным реляционным БД посвящена и работа [6] с акцентом в проектировании на ER-диаграммы и CASE-технологию.

В публикации [17] четко методически изложены отдельные теоретические аспекты баз данных (хранилища данных, параллельные БД).

Работа [3] носит обзорно-теоретический характер. Она полезна скорее преподавателям и научным сотрудникам для систематизации имеющихся знаний по базам данных. Начальное изучение вопросов, связанных с БД, с помощью такой книги проблематично.

Работы [3, 4, 6, 17] не могут, по нашему мнению, претендовать на роль системного изложения материалов по базам данных. К «системным» работам следует отнести публикации [1, 2, 5].

В работе [5], отличающейся хорошим терминологическим сопровождением, изложено современное состояние теории баз данных. В то же время, на наш взгляд, порядок изложения методически

недостаточно продуман, а прикладной материал практически отсутствует.

Наиболее фундаментальной в теоретическом плане является работа [2], рассчитанная на разработку — с использованием ER-диаграмм — концептуальной и логической моделей, в том числе в рамках хранилища данных.

В наибольшей мере требованию системности изложения материала удовлетворяет работа [1]. Вместе с тем она характеризуется, по нашему мнению, недостаточной системностью изложения. Так, описания СУБД не иллюстрируются примерами компьютерной реализации БД. Прикладная часть работы явно не связана с обсуждаемыми вопросами в теоретической части, что делает эти части фактически автономными.

В предлагаемом учебнике систематически изложено современное представление БД, включая такие новые и перспективные направления, как объектно-ориентированные и распределенные базы данных, автоматизация проектирования БД, хранилища данных, постреляционные БД.

Работа состоит из введения, 15 глав, заключения, и приложений. В главах 1—13 рассмотрены теоретические вопросы, главы 14, 15 посвящены прикладным вопросам реализации баз данных.

Во введении показана потребность в систематизации и структуризации сведений о современных БД, представленных в различных источниках, указаны причины написания настоящей работы.

Главы 1—4 посвящены общим положениям, связанным с базами данных.

В главе 1 приводится система определений, среди которых основными являются термины «данные», «информация», «знания» и их соотношения. Вводится понятие «база данных — БД», определяются статическая и динамическая разновидности БД.

Дается классификация баз данных и систем управления базами данных (СУБД) как оболочек, стандартных программных продуктов, на основе которых реализуются базы данных.

В главе 2 описаны основные принципы и подходы создания, использования и функционирования БД. Сформулированы требования, предъявляемые к базам данных. Основными требованиями являются быстродействие (динамические БД) и целостность данных (статические БД).

Рассмотрена блок-схема БД, в которой выделены основные составляющие: собственно база данных, интерфейс пользователя, алгоритм приложения.

Показано, что при рассмотрении БД следует выделить три основные процедуры: создание (проектирование) БД; использование (эксплуатация) БД; функционирование БД (взаимодействие составляющих).

Описана методология использования (с процедурой запроса).

В методологии проектирования определена суть традиционного (статические БД) и современного (динамические БД) подходов к построению баз данных, описаны особенности создания как статических, так и динамических баз данных. Рассмотрена суть основных моделей данных (типов структур) БД.

Описаны этапы проектирования, в том числе при построении концептуальной, логической и физической моделей. Определена цель и место процесса нормализации при создании статической базы данных.

Отмечено, что в соответствии в выделенными в БД процедурами, в теории баз данных, в свою очередь, можно выделить три составляющие: теорию построения БД; теорию использования БД; теорию функционирования БД. Указано, что наиболее полно все три составляющие теории развиты в полном объеме только для реляционных БД.

В главе 3 описана теория БД безотносительно к используемой модели данных. Рассмотрены основные варианты математического представления данных (модели): с помощью таблиц; с помощью аппарата теории графов; с использованием овал-диаграмм и ER-диаграмм (диаграмм «сущность—связь»). Указаны типы отношений между элементами данных. Отмечено, что перечисленные модели слабо ориентированы на использование компьютеров при проектировании БД.

Для автоматизации проектирования как небольших, так и сверхбольших БД предназначена CASE-технология.

В ней выделены и кратко описаны методы построения ER-диаграмм (собственно баз данных), диаграмм потоков связей (интерфейсы пользователей) и диаграмм переходов состояний (алгоритмов приложений).

Реализация CASE-технологии осуществляется CASE-средствами, которые могут быть встроены в СУБД (например, Oracle) или выполняться отдельно (например, CASE*Dictionary, CASE*Designer, CASE*Generator). Дана краткая характеристика автономных CASE-средств.

Глава 4 посвящена теории реляционных БД. Уточнены понятия «отношение», «ключ», являющиеся основными в реляционной модели данных.

Для **теории создания БД** использован математический аппарат реляционной алгебры (РА) и реляционного исчисления (РИ), отражающий процессы преобразования таблиц и нормализации данных. Описаны математические операции в РА и РИ. Приведены система аксиом РА для F- и MV-зависимостей между элементами данных и пример процедуры нормализации (от первой до пятой нормальных форм) при создании концептуальной и логической моделей. Показана связь РА и РИ как взаимодополняющих математических методов.

Рассмотрены понятия «безопасность», «целостность», «защита» данных, проанализированы источники нарушения целостности и варианты защиты данных. Описана суть хранилищ данных и их организация.

Теория использования БД при создании запросов опирается на РА и РИ. Показано, что РА «выходит» на алгоритмические языки программирования, которые неудобны для пользователей-непрофессионалов (ПН) в программировании. В то же время РИ является основой для декларативных языков программирования SQL, служащего средством связи между различными реляционными СУБД, и QBE (запрос по примеру). Последний является визуальным языком, понятным ПН и потому широко используемым в СУБД (например, Access). Рассмотрены теоретические предпосылки оптимизации запросов.

В **теории функционирования БД** рассмотрено понятие «транзакция» как основной программный элемент обеспечения целостности данных в статических БД. Под транзакцией понимается как входное сообщение (команда) на изменение (обновление) данных, так и процесс обновления. Показаны особенности работы транзакций в монопольном и многопользовательском режимах. В последнем случае говорят об одновременном доступе (синхронизации доступа) различных пользователей. Рассмотрены различные варианты синхронизации (блокировки, отложенные изменения, многоверсионность БД). Детально рассмотрен процесс блокировки как наиболее часто применяемый. Описан процесс восстановления данных при различного рода сбоях в БД.

Главы 5—10 посвящены прикладным вопросам централизованных (локальных) баз данных.

В главе 5 рассмотрены терминология, процедуры построения и использования реляционных БД. Подробно рассмотрен язык SQL при использовании его в обеих перечисленных процедурах. Представлены три разновидности языка SQL: интерфейсный, вложенный и встроенный. Описан пример использования языка QBE.

Глава 6 посвящена сетевым и иерархическим моделям данных с их специфической терминологией. Рассмотрена логическая структура, особенности программной реализации, в том числе создания и использования БД.

Глава 7 посвящена объектно-ориентированным базам данных (ООБД). Рассмотрена суть объектно-ориентированного подхода, объектно-ориентированного проектирования и объектно-ориентированного программирования, состояние и перспективы развития ООБД. Описана многомерная модель данных (как основа построения собственно ООБД), разновидности ее реализации: многомерная (MOLAP), реляционная (ROLAP), гибридная (HOLAP). Определено понятие «киоск (магазин, витрина) данных». Рассмотрены элементы структуры и их связи для многомерной модели, специфическая терминология этой разновидности моделей данных на примерах ООБД POET и CACHE. Показана связь ООБД с реляционными БД. Рассмотрены перспективы развития ООБД.

В главе 8 обсуждаются две разновидности «промежуточной» объектно-реляционной модели данных: гибридные, расширенные объектно-реляционные БД. Указана их специфика. Представлен программный продукт Delphi, на котором реализована гибридная ОРБД.

Глава 9 посвящена сравнению свойств моделей данных, описанных в главах 5—9. Рассмотрены варианты взаимопреобразования моделей данных. Описывается процесс выбора СУБД, состоящий из выбора модели данных и СУБД в рамках выбранной модели. Приводятся сравнительные характеристики ряда реляционных СУБД как наиболее часто используемых в прикладных целях. Обсуждается физическая модель БД (физическая БД), что важно при рассмотрении сетевой и в еще большей степени — иерархической моделей данных. Рассмотрены методы записи (размещения) и доступа (поиска) данных в памяти компьютера. Отмечено, что в суперЭВМ чаще всего используются быстродействующие иерархические СУБД, тогда как в персональных компьютерах — реляционные СУБД.

Главы 10—13 посвящены распределенным базам данных (РБД).

В главе 10 рассмотрены общие характеристики таких систем. Сформулированы новые требования, предъявляемые к вновь проектируемым БД и вызвавшие потребность в РБД. Дается определение РБД, ее состав и описание работы. Выделяются основные разновидности структуры РБД: одноранговая (наиболее сложная по структуре), клиент—сервер и смешанная. Приведен пример-описание одноранговой РБД, дана сравнительная характеристика стратегий хра-

нения данных в РБД. Описаны особенности и варианты структуры клиент—сервер. Рассмотрены особенности реализации этой структуры в СУБД Paradox и InterBase (в рамках программного продукта Delphi).

В последующих главах рассмотрены процедуры создания, функционирования и использования одноранговой РБД.

В главе 11 описаны особенности создания РБД. Рассмотрены вопросы обеспечения целостности, фрагментации и локализации данных как специфические и дополнительные (по сравнению с централизованной БД) этапы проектирования. Введены понятия «однородные РБД» (при использовании в локальных БД распределенной БД одинаковых, чаще реляционных, моделей данных), «неоднородные РБД» (при использовании в локальных БД различных моделей данных). Рассмотрена математическая модель процесса интеграции локальных БД, связанная с преобразованием данных из одной модели данных в другую. Показана необходимость сопоставления наборов типов данных и процедур в языках описания и манипулирования данными для различных СУБД.

Рассмотрены два варианта преобразования: построение универсальной промежуточной модели, чаще используемой в неоднородных РБД, и системы драйверов для попарного соединения СУБД (в однородных РБД с локальными реляционными БД) с программной реализацией через продукт ODBC. Приведены примеры связей реляционных СУБД (соответствие типов данных, использование языка SQL и промежуточного программного продукта ODBC).

Рассмотрены процессы преобразования «реляционная — сетевая», «реляционная — иерархическая» модели данных.

В главе 12 рассматриваются особенности и методы обеспечения одновременного доступа (разновидности блокировки, многоверсионность БД, отложенные (кэшированные) изменения данных). Описаны варианты защиты данных, специфика резервного копирования и восстановления данных. Обсуждены способы оптимизации запросов в РБД.

В главе 13 обсуждаются вопросы Web-публикаций баз данных в Internet. Определяется понятие «публикация», рассматриваются основы используемого для публикации языка программирования HTML и приводится прикладной пример с использованием программного продукта Delphi.

Главы 14, 15 освещают вопросы реализации баз данных с помощью различных СУБД. Рассмотрены общие вопросы реализации баз данных и традиционный подход на примере базы данных «Учебный процесс».

Обсуждаются два варианта реализации. В локальном варианте используется СУБД Access, для режима клиент—сервер — СУБД InterBase. В обоих вариантах процесс проектирования и реализации рассматривается в полном объеме.

Глава 15 посвящена современному подходу с использованием СУБД InterBase на примере процедуры приема на работу. Компьютер выдает руководителю решения-советы, которые могут быть руководителем приняты или отвергнуты (с корректировкой решений). Процедура проектирования по-прежнему рассматривается в полном объеме. Описываются как локальный, так и удаленный варианты режима клиент—сервер, вопросы реализации одно- и многоуровневой структуры с использованием программного продукта Delphi.

В заключении подводятся итоги изучения баз данных.

В приложении 1 приведены основные характеристики БД «Учебный процесс». Приложения 2 и 3 содержат программы, используемы при работе с СУБД InterBase в рамках приложения Delphi.

Глава 1

Общие сведения

Показана «табличная» суть базы данных на примере процесса автоматизации табличных расчетов. Введены необходимые понятия и определения. Приведена классификация баз данных и систем управления базами данных. Отмечено, что теория баз данных имеет три основных составляющих: теория создания, теория использования и теория работы (функционирования) баз данных. С позиций реализации баз данных выделено три составляющих: реализация собственно базы данных (система связанных таблиц), интерфейса пользователя и алгоритма приложения.

1.1. БАЗА ДАННЫХ И АВТОМАТИЗАЦИЯ ТАБЛИЧНЫХ РАСЧЕТОВ

Считается, что понятие «база данных» (БД), а тем более — «система управления базами данных» (СУБД) достаточно сложно в усвоении. Оно значительно упрощается, если понять «физическую сущность» процессов, происходящих в изучаемом программном продукте.

Следует помнить, что многие программные продукты являются средством автоматизации соответствующих ручных процедур, которые просты в понимании. В частности, база данных (при так называемом традиционном подходе к проектированию) служит инструментом автоматизации расчетов, в которых входные и выходные данные представлены в виде системы таблиц с большим числом строк и столбцов в каждой.

Такие расчеты имеют место в процессах проектирования различных процессов, технологий, устройств, блоков; в управлении производством; в курсовых и дипломных проектах.

Подтвердим сказанное простым конкретным примером. Начнем с ручного расчета.

Пример 1.1.

Задание. Имеется склад материалов (М1, М2, М3), снабжающий производство, выпускающее изделия И1, И2. Каждое из изделий

состоит из деталей Д1, Д2 и Д3, количество которых в изделиях (входимость) известно. Известны и нормы расхода материалов на каждую деталь. Задан — на протяжении месяца — ежедневный план выпуска изделий, который склад должен обеспечить материалами. Длительность технологического цикла (производства) изделий $t_ц$ — 3 дня. Рекомендуется создать страховой запас материалов на складе на один день. Склад периодически заказывает материалы поставщикам. Время τ выполнения заказа r по первому и третьему материалам — 2 дня, по второму — 1 день. Поставки материалов производятся по первому и третьему материалам — каждый третий день, по второму — каждый четвертый день.

Необходимо ежедневно на протяжении месяца определить (решить задачи):

1) запуск материалов под план производства;
2) заказы на материалы;
3) поставки материалов;
4) движение материалов на складе по дням.

Исходные данные получены из различных источников.

Решение. Поскольку исходные данные получены из разных источников, целесообразно упорядочить их в табличной форме. Вид таблиц может быть различным.

Пусть данные упорядочены в виде системы таблиц: «Материалы», «Детали», «Изделия», «Нормы» (расхода), «Входимость», «План» (выпуска изделий) (табл. 1.1—1.6).

Таблица 1.1

Материалы

Шифр материала	Название материала	Единица измерения
М1	Сталь	кг
М2	Чугун	кг
М3	Железо	кг

Таблица 1.2

Детали

Шифр детали	Название детали	Единица измерения
Д1	Втулка	шт.
Д2	Фланец	шт.
Д3	Палец	шт.

Таблица 1.3

Изделие

Шифр изделия	Наименование изделия	Единица измерения
И1	Изделие 1	шт.
И2	Изделие 2	шт.

Таблица 1.4

Нормы

Шифр материала	Шифр детали	Единица измерения	Норма расхода
М1	Д1	кг/шт	2,1
М1	Д2	кг/шт	1,8
М2	Д1	кг/шт	3,4
М2	Д3	кг/шт	4,3
М3	Д2	кг/шт	1,5
М3	Д3	кг/шт	3,7

Таблица 1.5

Входимость

Шифр детали	Шифр изделия	Единица измерения	Количество
Д1	И1	шт/изд	7
Д2	И1	шт/изд	9
Д2	И2	шт/изд	8
Д3	И2	шт/изд	5

Таблица 1.6

План

Дата	Шифр изделия	Количество изделий
1	И1	7
2	И1	11
3	И1	8
4	И1	9

Продолжение табл. 1.6

Дата	Шифр изделия	Количество изделий
5	И1	12
6	И1	13
7	И1	9
8	И1	8
9	И1	7
10	И1	9
11	И1	14
12	И1	11
13	И1	8
14	И1	4
15	И1	7
16	И1	5
17	И1	8
18	И1	9
19	И1	10
20	И1	14
21	И1	17
22	И1	12
23	И1	11
24	И1	10
25	И1	8
1	И2	5
2	И2	12
3	И2	8
4	И2	7

Продолжение табл. 1.6

Дата	Шифр изделия	Количество изделий
5	И2	4
6	И2	3
7	И2	15
8	И2	4
9	И2	8
10	И2	7
11	И2	4
12	И2	20
13	И2	25
14	И2	14
15	И2	11
16	И2	10
17	И2	15
18	И2	5
19	И2	7
20	И2	10
21	И2	8
22	И2	4
23	И2	12
24	И2	15
25	И2	18

Структура (совокупность элементов и их связей) приведенных таблиц является линейной.

Однако система исходных таблиц может быть представлена иначе. Например, вместо таблиц «Детали» и «Нормы» может быть построена одна таблица «Детали—Нормы» (табл. 1.7).

Таблица 1.7

Детали—Нормы

Шифр детали	Название детали	Единица измерения	Норма расхода материалов (М1, М2, М3)
Д1	Втулка	шт.	2,1; 3,4; 0
Д2	Фланец	шт.	1,8; 0; 1,5
Д3	Палец	шт.	0; 4,3; 3,7

Нетрудно видеть, что в последнем столбце таблицы «Детали—Нормы» используются списки. Данные такого рода носят название неатомарных. Все остальные данные таблиц являются атомарными.

Возможен и другой вариант организации таблиц «Материалы», «Детали», «Нормы» в виде одной таблицы «Материалы—Детали—Нормы» (табл. 1.8).

Таблица 1.8

Материалы—Детали—Нормы

Шифр материала	Название материала	Единица измерения	Детали			Норма расхода
			Шифр детали	Название детали	Единица измерения	
М1	Сталь	кг	Д1	Втулка	шт.	2,1
		кг	Д2	Фланец	шт.	1,8
		кг	Д3	Палец	шт.	0
М2	Чугун	кг	Д1	Втулка	шт.	3,4
		кг	Д2	Фланец	шт.	0
		кг	Д3	Палец	шт.	4,3
М3	Железо	кг	Д1	Втулка	шт.	0
		кг	Д2	Фланец	шт.	1,5
		кг	Д3	Палец	шт.	3,7

В последней таблице имеет место «таблица в таблице». Таблицы «Детали—Нормы» и «Материалы—Детали—Нормы» обладают нелинейной структурой.

Заметим, что в ручном режиме переход от таблиц с линейной структурой к таблицам с нелинейной структурой и наоборот ника-

ких затруднений не вызывает. Структуру исходных таблиц определяет пользователь, исходя из удобства работы.

Отметим также уязвимость (нарушение целостности данных) «ручных» таблиц с большим числом столбцов и строк: при заполнении таблиц в столбец с числовыми данными по ошибке могут быть записаны символьные данные.

Чтобы уменьшить число таких ошибок, при заполнении подобных таблиц часто накладывают линейку на выбранную строку.

Одновременно акцентируем внимание на технологии формирования системы исходных таблиц:

- создание структуры полей («шапки») таблицы;
- определение столбцов для связи таблиц в последующих вычислениях (например, столбец «Шифр материала» для таблиц «Материалы» и «Нормы»), хотя явно связи не задаются;
- заполнение таблиц данными.

Снова отметим, что по ошибке в столбце «Нормы» может быть вписано значение, отсутствующее в одноименном столбце таблицы «Материалы» (нарушение ссылочной целостности).

Для получения результатов расчета можно использовать систему таблиц с любой структурой. Для определенности воспользуемся таблицами с линейной структурой. Приступим к решению задач примера.

Задача 1. *Специфицированной потребностью (нормой)* называют количество M_{ijk} материала M_i, необходимого для одного изделия $И_k$, если учитывать только деталь $Д_j$

$$M_{ijk} = H_{ij} B_{jk}, \tag{1.1}$$

где *H и B — норма и входимость, соответственно.*

Для материала М1, детали Д1 и изделия И1

$$M_{111} = 2{,}1 \cdot 7 = 14{,}7. \tag{1.2}$$

Количество материала M_{ik} для одного изделия

$$M_{ik} = \sum_{j=1}^{3} H_{ij} B_{jk}. \tag{1.3}$$

Для первого материала и первого изделия

$$M_{11} = 2{,}1 \cdot 7 + 1{,}8 \cdot 9 = 30{,}9. \tag{1.4}$$

В базах данных вычисления вида (1.3) называют агрегированными по сравнению с выражением (1.1). Функцией агрегации здесь является сумма.

Нетрудно видеть, что вычисления (1.1)—(1.4) являются фактически матричными вычислениями вида

$$M = H \cdot B,$$

которые в числовой форме имеют вид

$$\begin{vmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \\ M_{31} & M_{32} \end{vmatrix} = \begin{vmatrix} 2{,}1 & 1{,}8 & 0 \\ 3{,}4 & 0 & 4{,}3 \\ 0 & 1{,}5 & 3{,}7 \end{vmatrix} \cdot \begin{vmatrix} 7 & 0 \\ 9 & 8 \\ 0 & 5 \end{vmatrix} = \begin{vmatrix} 30{,}9 & 14{,}4 \\ 23{,}8 & 21{,}5 \\ 13{,}5 & 30{,}5 \end{vmatrix}.$$

Если ежедневный план выпуска $P[t_l] = \{P_k[t_l]\}$, $l = \overline{1, L}$ — текущий день, то ежедневный запуск (потребность в материалах, называемая *потребностью в укрупненной номенклатуре*), в матричной форме имеет вид

$$M[t_l] = M \cdot P[t_l], \tag{1.5}$$

а в числовом виде для первого дня

$$M[1] = \begin{vmatrix} M_1[1] \\ M_2[1] \\ M_3[1] \end{vmatrix} = \begin{vmatrix} 30{,}9 & 14{,}4 \\ 23{,}8 & 21{,}5 \\ 13{,}5 & 30{,}5 \end{vmatrix} \cdot \begin{vmatrix} 7 \\ 5 \end{vmatrix} = \begin{vmatrix} 288{,}3 \\ 274{,}1 \\ 247{,}0 \end{vmatrix}. \tag{1.6}$$

Перейдем к решению других задач. Задачи 2 и 3 отличаются лишь сдвигом во времени на величину r, поэтому рассмотрим решение лишь третьей задачи.

Задача 3. Величину поставки можно рассчитывать различными способами. Произведем расчет, исходя из средней ежедневной потребности в материалах МС, необходимой для выполнения плана

$$MC = \sum_{t=1}^{T} M[t] / T, \tag{1.7}$$

где T — количество дней в месяце (в данном случае — 25). Операция (1.7) в базах данных также относится к операциям агрегации с усреднением в качестве функции агрегации.

Очевидно

$$MC = \begin{vmatrix} M_1 \\ M_2 \\ M_3 \end{vmatrix}. \qquad (1.8)$$

Тогда величина поставки

$$П = MC \cdot (t_д + 1) \qquad (1.9)$$

или

$$П = \begin{vmatrix} 1327{,}36 \\ 1781{,}17 \\ 1309{,}08 \end{vmatrix}. \qquad (1.10)$$

Ежедневные поставки для первого и третьего материалов ($i = 1$, $i = 3$)

$$П_1[t] = \begin{cases} П_1, & \text{если величина } t \text{ времени кратна трем;} \\ 0, & \text{в остальных случаях,} \end{cases} \qquad (1.11)$$

для второго материала

$$П_2[t] = \begin{cases} П_2, & \text{если величина } t \text{ времени кратна четырем;} \\ 0, & \text{в остальных случаях.} \end{cases} \qquad (1.12)$$

Задача 4. Расчет запасов $З(t)$ материалов ведется по формуле

$$З(t) = З(t-1) + П[t] - М[t], \qquad (1.13)$$

где (t), $[t]$ — моменты и интервалы времени, соответственно. Иными словами, запасы $З(t)$ отличаются от запасов в начале дня. Отметим, что новый запас данного дня становится старым запасом следующего дня.

Очевидно

$$\begin{vmatrix} З_1(1) \\ З_2(1) \\ З_3(1) \end{vmatrix} = \begin{vmatrix} З_1(0) + П_1(1) - М_1(1) \\ З_2(0) + П_2(1) - М_2(1) \\ З_3(0) + П_3(1) - М_3(1) \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} - \begin{vmatrix} 288{,}3 \\ 274{,}1 \\ 247{,}0 \end{vmatrix} = \begin{vmatrix} -288{,}3 \\ -274{,}1 \\ -247{,}0 \end{vmatrix}. \qquad (1.14)$$

В общем виде

$$З(1) = З(0) + П[1] - М[1],$$
$$З(2) = З(1) + П[2] - М[2]. \qquad (1.15)$$

На этом ручной расчет закончен.

Очевидно, что при небольшом количестве таблиц, столбцов и строк в каждой из них возможен ручной расчет, технология которого не вызывает особых затруднений.

Однако часто задачи содержат более десяти таблиц, в которых свыше 15 столбцов и 20 000 строк. При этом исходные данные могут оперативно изменяться и требуется постоянная корректировка результатов. Электронные таблицы (Excel) здесь использовать неудобно, поскольку связь и целостность данных таблиц обеспечить сложно, да и количество строк в таблице не должно превышать 16 000.

В этом случае без баз данных не обойтись.

На первый взгляд, технология работы с БД не должна вызывать осложнений. По-прежнему следует:

- сформировать структуру («шапку») для всех таблиц;
- определить связи таблиц;
- заполнить их данными.

Для сравнения рассмотрим результаты компьютерной реализации примера 1.1 с использованием, в частности, СУБД Access.

Компьютерная таблица «Детали» (рис. 1.1) полностью совпадает с ручной таблицей «Детали» (см. табл. 1.2).

Схема связей таблиц, созданная до работы БД, показана в явном виде на рис. 1.2.

Следует отметить, что связи в СУБД Access могут быть заданы и в процессе формирования запросов (выполнения расчетов). Однако предпочтительнее делать это до расчетов, что надежно обеспечит целостность данных:

1) при заполнении подчиненных таблиц компьютер не внесет данные с внешними ключами, отсутствующими в главной таблице (ссылочная целостность);

Шифр детали	Название детали	Единица измерения
Д1	Втулка	шт.
Д2	Фланец	шт.
Д3	Палец	шт.

Рис. 1.1. Таблица «Детали базы данных» в СУБД Access

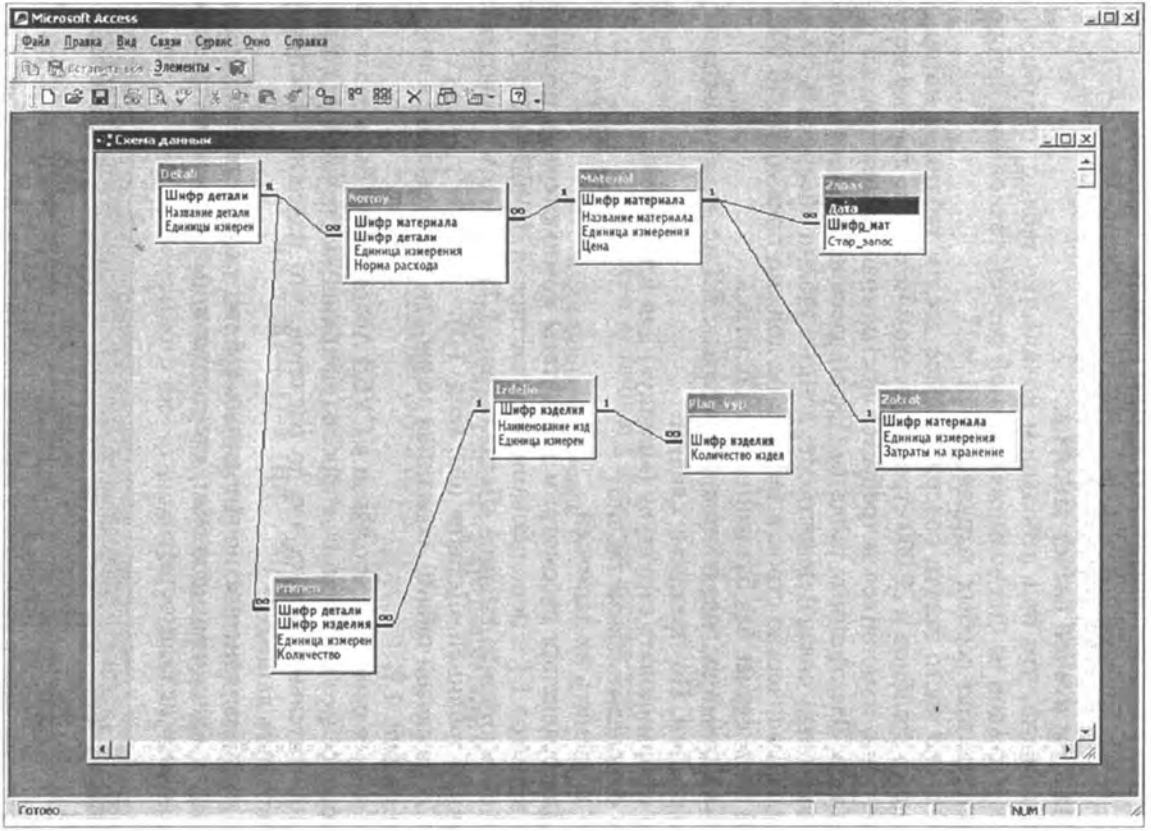


Рис. 1.2. *Схема связей таблиц*

2) если не обеспечить ссылочную целостность данных до использования БД, то при ошибочном заполнении компьютер (в процессе запроса) может и не создать связи между таблицами.

Таким образом, выявилось первое ограничение работы с БД.

Из рис. 1.2 видно также, что таблицы имеют линейную структуру.

Заполнение БД данными может быть осуществлено прямо в таблицу (рис. 1.1) или через форму (рис. 1.3), как аналог наложения линейки на строку таблицы при вводе данных в ручном режиме. Вид форм может быть различным. В частности, он может полностью копировать расположение столбцов ручного документа, из которого заимствуются данные для базы данных.

Выходные данные (результаты расчетов) оформляются в виде запросов или отчетов (если к запросам необходимы пояснения).

Сформируем запрос на визуальном языке QBE, не требующем знания программирования.

Запрос, представленный на рис. 1.4, реализует выражение (1.1).

Хотя запрос строится с применением языка QBE, компьютер автоматически формирует оператор на языке SQL, являющимся основным языком работы баз данных.

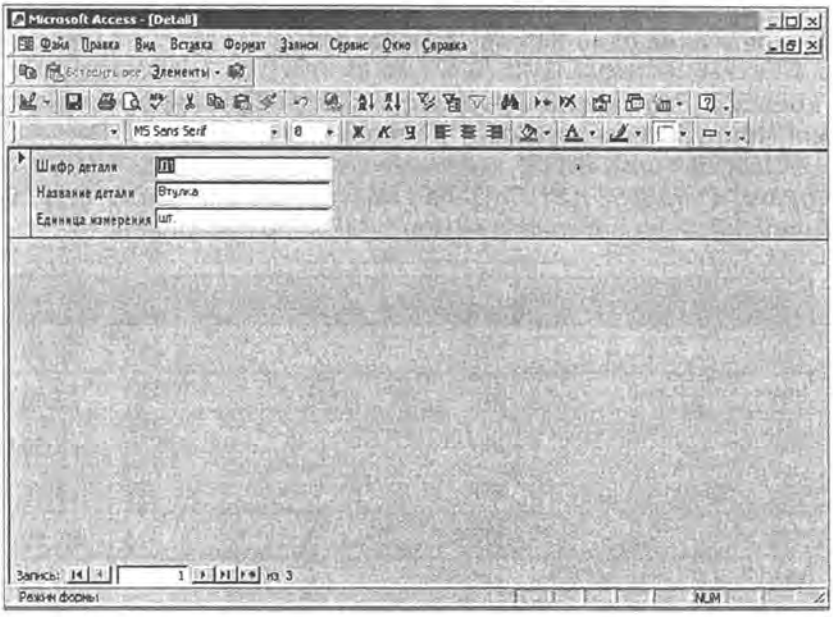


Рис. 1.3. Форма для заполнения таблицы «Детали»

Шифр изделия	Шифр материала	Шифр детали	Расход 1
И1	М3	Д2	13,50
И1	М1	Д2	16,20
И1	М2	Д1	23,80
И1	М1	Д1	14,70
И2	М3	Д3	18,50
И2	М2	Д3	21,50
И2	М3	Д2	12,00
И2	М1	Д2	14,40

Рис. 1.4. Расход (неагрегированный) материала на одно изделие

```
SELECT Primen.[Шифр изделия], Material.[Шифр материала],
[Normy]![Норма расхода]*[Primen]![Количество] AS Расход1,
Primen.[Шифр детали]
FROM Material INNER JOIN (Normy INNER JOIN Primen ON
Normy.[Шифр детали] = Primen.[Шифр детали]) ON Material.[Шифр
материала] = Normy.[Шифр материала]
ORDER BY Primen.[Шифр изделия];
```

Запрос на рис. 1.5 характеризует агрегированные потребности в в материалах на одно изделие (выражение (1.3)).

```
SELECT [Primen].[Шифр изделия], [Material].[Шифр материала],
Sum([Normy]![Норма расхода]*[Primen]![Количество]) AS Расход2
FROM Material INNER JOIN (Normy INNER JOIN Primen ON
[Normy].[Шифр детали]=[Primen].[Шифр детали]) ON
[Material].[Шифр материала]=[Normy].[Шифр материала]
```

Шифр изделия	Шифр материала	Расход 2
И1	М1	30,90
И1	М2	23,80
И1	М3	13,50
И2	М1	14,40
И2	М2	21,50
И2	М3	30,50

Рис. 1.5. Расход (агрегированный) материала на одно изделие

Дата	Шифр материала	Название материала	Количество материала
1	M1	Сталь	288,30
1	M2	Чугун	274,10
1	M3	Железо	247,00
2	M1	Сталь	512,70
2	M2	Чугун	519,80
2	M3	Железо	514,50

Рис. 1.6. Расход материалов на выполнение плана (запуск)

```
GROUP BY [Primen].[Шифр изделия], [Material].[Шифр материала]
ORDER BY [Primen].[Шифр изделия];
```

Запрос (рис. 1.6) на основе выражения (1.5) позволяет видеть ежедневные плановые затраты материалов.

Запрос на рис. 1.7 определяет поставки в соответствии с выражениями (1.11)—(1.12)

```
SELECT Zapusk.Дата, Zapusk.[Шифр материала],
IIf([Zapusk]![Шифр материала]=»M2",IIf(([Zapusk]![Дата] Mod
4)=0,[Postavka_sum]![Среднее]*4,0),IIf(([Zapusk]![Дата] Mod
3)=0,[Postavka_sum]![Среднее]*3,0)) AS Поставка, IIf([Zapusk]![Шифр
материала]=»M2",IIf(([Zapusk]![Дата] Mod
3)=0,[Postavka_sum]![Среднее]*4,0),IIf(([Zapusk]![Дата] Mod
3)=1,[Postavka_sum]![Среднее]*3,0)) AS Заказы
```

Дата	Шифр материала	Поставка	Заказ
1	M1	0,00	1327,36
1	M2	0,00	0,00
1	M3	0,00	1309,08
2	M1	0,00	0,00
2	M2	0,00	0,00
2	M3	0,00	0,00
3	M1	1327,36	0,00
3	M2	0,00	1781,17
3	M3	1309,08	0,00
4	M1	0,00	1327,36
4	M2	1781,17	0,00
4	M3	0,00	1309,08

Рис. 1.7. Поставка материалов

Дата	Шифр материала	Название материала	Стар_запас	Поставка	Количество материала	Нов_запас	Дата
1	M1	Сталь	0,00	0,00	288,30	-288,30	2
1	M2	Чугун	0,00	0,00	274,10	-274,10	2
1	M3	Железо	0,00	0,00	247,00	-247,00	2
2	M1	Сталь	-288,30	0,00	512,70	-801,00	3
2	M2	Чугун	-274,10	0,00	519,80	-793,90	3
2	M3	Железо	-247,00	0,00	514,50	-761,50	3
3	M1	Сталь	-801,00	1327,36	362,40	163,96	4
3	M2	Чугун	-793,90	0,00	362,40	-1156,30	4
3	M3	Железо	-761,50	1309,08	352,40	195,58	4
4	M1	Сталь	163,96	0,00	378,90	-214,94	5
4	M2	Чугун	-1156,30	1781,17	364,70	260,17	5
4	M3	Железо	195,58	0,00	335,00	-139,42	5

Рис. 1.8. Изменение запасов материалов на складе

```
FROM Zapusk INNER JOIN Postavka_sum ON Zapusk.[Шифр материала] = Postavka_sum.[Шифр материала];
```

Нетрудно видеть присутствие в SQL-запросе программного оператора условного перехода

```
Iif([Zapusk]![Шифр материала]=»M2"...,
```

вводимого дополнительно в процессе запроса. Это говорит об ограниченных возможностях языка QBE.

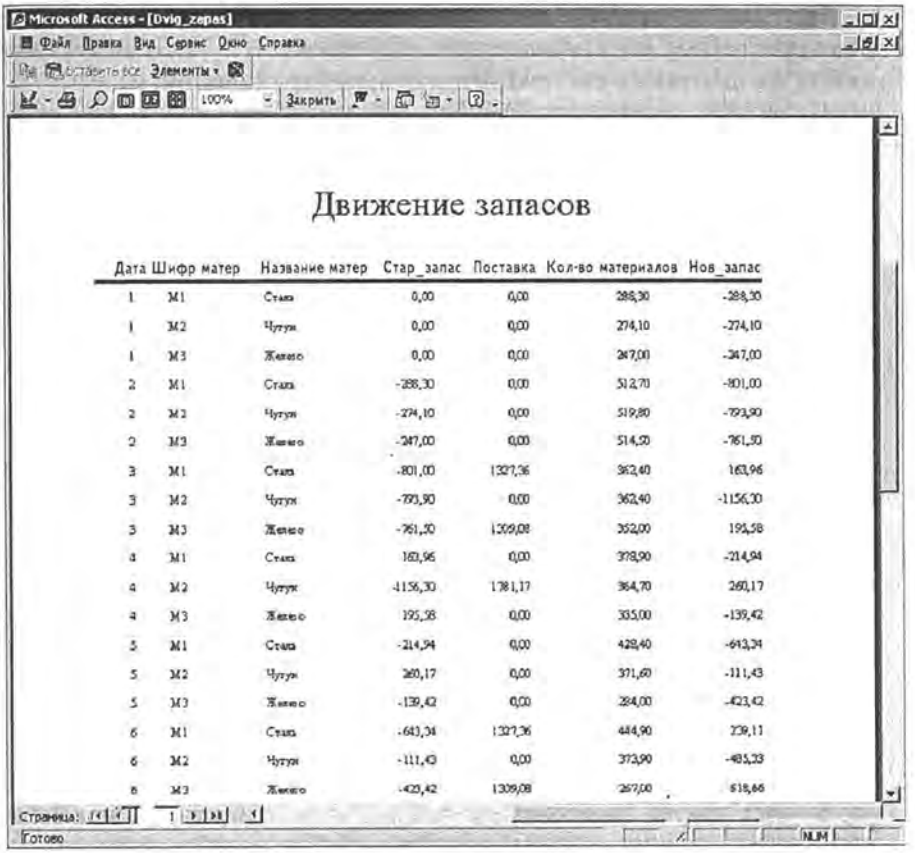
Запрос на рис. 1.8 изменения запасов материалов формируется уже из нескольких предварительных запросов. Автоматизация их выполнения может быть достигнута либо использованием макросов (как в данном случае), либо написанием программы на языке VBA.

Таким образом, еще раз подтвержден факт, что за простоту языка QBE приходится «платить» резким ограничением его возможностей.

На рис. 1.9 показан отчет об изменении запасов материалов. Нетрудно видеть его отличие от запроса (рис. 1.8), на основе которого построен отчет.

На рис 1.10 представлены графики изменения запасов материалов, построенные на основе соответствующего запроса. Отметим, что они несут больше информации, чем таблицы. Более того, можно воспользоваться графиками для принятия решений о выборе величины поставок в выражении (1.10).

Таким образом, при традиционном подходе имеется прямая аналогия процедур ручного расчета с помощью систем таблиц и компьютерного расчета с использованием БД. Хотя при современном подходе эта аналогия имеет завуалированный вид, она позволяет лучше понять процессы, имеющие место в БД.



Движение запасов

Дата	Шифр матер	Название матер	Стар_запас	Поставка	Кол-во материалов	Нов_запас
1	M1	Сталь	0,00	0,00	288,30	-288,30
1	M2	Чугун	0,00	0,00	274,10	-274,10
1	M3	Железо	0,00	0,00	247,00	-247,00
2	M1	Сталь	-288,30	0,00	512,70	-801,00
2	M2	Чугун	-274,10	0,00	519,80	-793,90
2	M3	Железо	-247,00	0,00	514,50	-761,50
3	M1	Сталь	-801,00	1327,36	362,40	163,96
3	M2	Чугун	-793,90	0,00	362,40	-1156,30
3	M3	Железо	-761,50	1309,08	352,00	195,58
4	M1	Сталь	163,96	0,00	378,90	-214,94
4	M2	Чугун	-1156,30	1781,17	364,70	260,17
4	M3	Железо	195,58	0,00	335,00	-139,42
5	M1	Сталь	-214,94	0,00	428,40	-643,34
5	M2	Чугун	260,17	0,00	371,60	-111,43
5	M3	Железо	-139,42	0,00	284,00	-423,42
6	M1	Сталь	-643,34	1327,36	444,90	239,11
6	M2	Чугун	-111,43	0,00	373,90	-485,23
6	M3	Железо	-423,42	1309,08	267,00	618,66

Рис. 1.9. Отчет о движении материалов на складе

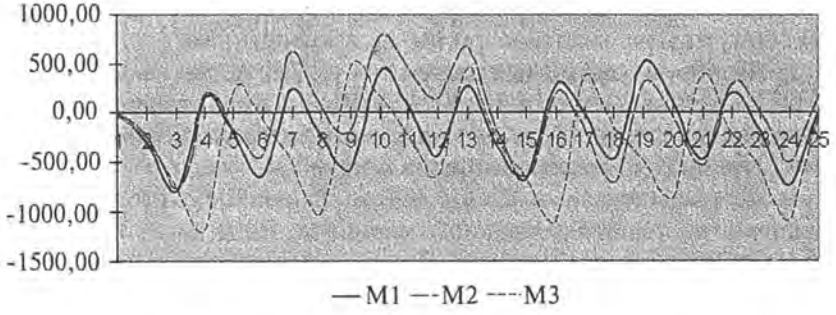


Рис. 1.10. График движения запасов материалов на складе

Напомним, что при ручных операциях особых сложностей не возникает. Они начинаются при обращении к СУБД, поскольку любая из них имеет систему структурных ограничений.

Структуры различных СУБД имеют как общие, так и специфические свойства. Общие структурные свойства определяет *модель данных (МД). Она не зависит от содержания конкретной БД и отвечает* на вопрос «Каковы общие структурные элементы и как они связаны между собой?».

Известны иерархические, сетевые, реляционные, объектно-ориентированные и объектно-реляционные МД. В настоящее время наиболее широко используются реляционные и объектно-реляционные (гибридные) БД. Имеется тенденция к переходу к объектно-ориентированным МД.

В реляционных МД структурными элементами являются таблицы, а связи между ними осуществляются через ключи. В таблицах выделяют строки (записи) и столбцы (поля), которые и участвуют в различных преобразованиях. Для реляционных МД характерны следующие ограничения.

1. Ячейка (пересечение столбца и строки) должна быть атомарна. В нее не допускается помещать данные в виде списков, подтаблиц и т. д. Атомарность достигается использованием так называемой первой нормальной формы.

2. Линейная структура таблиц. Если структура нелинейна, проводят соответствующее преобразование, называемое *нормализацией* (построение второй и третьей нормальных форм).

3. Отсутствует наследование таблиц: нельзя получить из какой-либо таблицы другую путем удаления одних и добавления новых полей. Можно лишь формировать новые таблицы-запросы.

Названные ограничения отсутствуют в объектно-ориентированных и расширенных объектно-реляционных моделях данных.

Различия СУБД одной модели данных (в частности, реляционных) могут иметь место по таким характеристикам.

1. По объему хранимых данных — СУБД Access — до 1 Гбайта, СУБД InterBase — до 10 Гбайт, СУБД Oracle — свыше 10 Гбайт. Другими характеристиками могут быть предельное количество столбцов, строк, количество символов в поле.

2. По назначению — СУБД Access, Paradox, FoxPro изначально предназначались для локального варианта, тогда как СУБД SyBase, Informix, SQL Server, Interbase, Oracle — для работы в сети (удаленного варианта).

3. По обеспечению целостности данных, т. е. противодействию внесения неверных данных (например, возраст 1 000 лет) с помо-

щью специальных программ-триггеров — в СУБД Access эти программы являются встроенными, тогда как в СУБД InterBase такие программы вводятся разработчиком БД.

4. По ориентации на уровень пользователя — СУБД Access предназначена прежде всего для начинающих пользователей, практически не знающих языков программирования. Для работы используется визуальный язык программирования QBE, который предполагает у СУБД наличие развитого интерфейса.

Вместе с тем возможности такой СУБД при использовании только языка QBE, как показано ранее, резко ограничиваются.

Так, в примере 1.1 реализация выражений (1.11) и (1.12) требует простейшего программирования (IIf <условие>; <результат, если условие выполнено>; <результат, если условие не выполнено>).

Сказанное относится и к выражениям (1.13)—(1.15), для реализации которых необходимо программирование либо при помощи макросов, либо с применением алгоритмического языка Visual Basic for Applications (VBA).

В то же время СУБД InterBase рассчитана на так называемых «продвинутых» пользователей, которые знакомы с языками программирования SQL и Object Pascal. Для этой СУБД реализация выражений вида (1.11)—(1.15) не вызывает затруднений.

Таким образом, прикладные примеры для начинающих пользователей проиллюстрированы на СУБД Access (гл. 15), а для «продвинутых» — на СУБД InterBase в среде программного продукта Delphi (гл. 15).

Возможно и далее перечислять ограничения СУБД, однако целесообразнее провести системное рассмотрение процесса работы с базами данных, опираясь на соответствующий набор строгих определений.

1.2. ДАННЫЕ, ИНФОРМАЦИЯ, ЗНАНИЯ

Любая задача обработки информации и принятия решений может быть представлена в виде схемы, показанной на рис. 1.11.

В качестве составных частей схемы выделяются информация (входная и выходная) и правила ее преобразования. Правила могут быть в виде алгоритмов, процедур и эвристических последовательностей.

Алгоритм — последовательность правил перехода от исходных данных к результату. Правила могут выполняться компьютером или человеком.

Данные — совокупность объективных сведений.

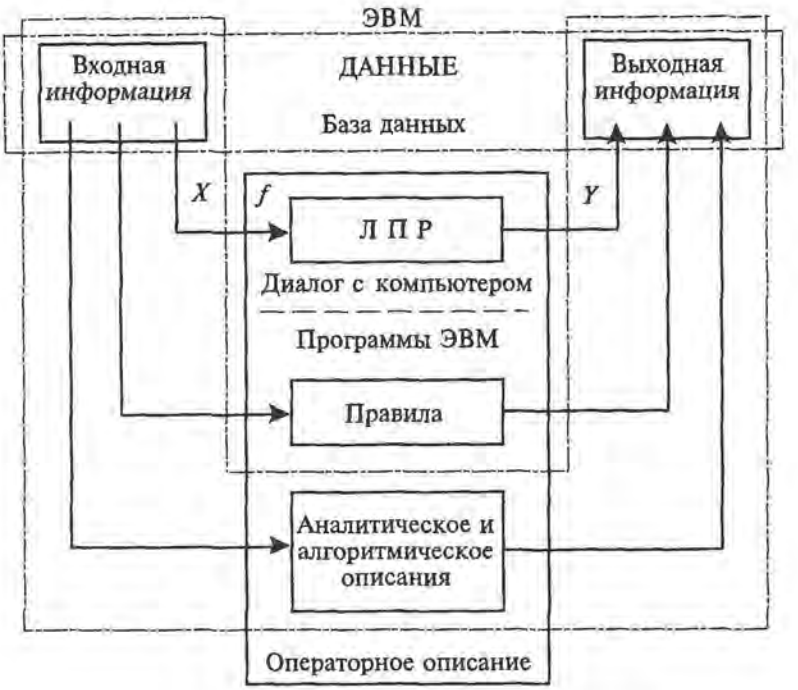


Рис. 1.11. Схема решения задач обработки информации и принятия решений:

x и *y* — входная и выходная информации;
f — внутреннее операторное описание

Информация — сведения, неизвестные ранее получателю информации, пополняющие его знания, подтверждающие или опровергающие положения и соответствующие убеждения. Информация носит субъективный характер и определяется уровнем знаний субъекта и степенью его восприятия. Информация извлекается субъектом из соответствующих данных.

Знания — совокупность фактов, закономерностей и эвристических правил, с помощью которых решается поставленная задача.

Последовательность операций обработки данных называют информационной технологией (ИТ). В силу значительного количества информации в современных задачах она должна быть упорядочена. Существует два подхода к упорядочению.

1. Данные связаны с конкретной задачей (технология массивов) — упорядочение по использованию. Вместе с тем алгоритмы более подвижны (могут чаще меняться), чем данные. Это вызывает необ-

ходимость переупорядочения данных, которые к тому же могут повторяться в различных задачах.

2. Другая, более широко используемая технология баз данных, — упорядочение по хранению.

Под **базой данных** (БД) понимают совокупность хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование для одного или нескольких приложений.

Существует две информационные технологии организации данных: технология массивов и технология баз данных.

В *технологии массивов* для каждого алгоритма приложения формируется своя система таблиц. Покажем это на простейшем примере.

Для вычисления по алгоритму $y = ax + b$, где a и b — константы, потребуются входная и выходная таблицы (табл. 1.9 и 1.10).

Таблица 1.9

Входная

Порядковый номер	a	b	x

Таблица 1.10

Выходная

Порядковый номер	a	b	x	y

Для приложения с алгоритмом $y = ax$ формируются новые таблицы (без столбца b), для функции $y = (ax + b)^2$ снова нужны дополнительные входная и выходная таблицы, хотя данные для вычислений уже имеются. Следовательно, существует избыточная информация.

Если использовать *технологию баз данных*, то для всех названных алгоритмов достаточно двух приведенных таблиц. Все необходимые преобразования данных осуществляются с помощью алгоритмов (программ) СУБД.

Таким образом, требуется минимальное количество данных, которые можно использовать для различных приложений (в данном примере — функций). Дублирование данных может проводиться лишь с позиций повышения надежности хранения данных в сверхбольших БД с объемом более 10 Гбайт.

Целью создания баз данных, как разновидности информационной технологии и формы хранения данных, является построение системы данных, не зависящих от принятых алгоритмов (программного обеспечения), применяемых технических средств и физического расположения данных в компьютере; обеспечивающих непротиворечивую и целостную информацию при нерегламентируемых запросах. БД предполагает многоцелевое ее использование (несколько пользователей, множество форм документов и запросов одного пользователя).

База знаний (БЗ) представляет собой совокупность БД и используемых правил, полученных от лиц, принимающих решения (ЛПР).

Наряду с понятием «база данных» существует термин **«банк данных»**, который имеет две трактовки.

1. В настоящее время данные обрабатываются децентрализованно (на рабочих местах) с помощью персональных компьютеров (ПК). Первоначально же использовалась централизованная обработка на больших ЭВМ, размещенных, как правило, в отдельном помещении — вычислительном центре (ВЦ). В ВЦ стекалась вся информация с периферийных устройств. В силу централизации базу данных называли банком данных и потому часто не делают различия между базами и банками данных.

2. Банк данных — база данных и система управления ею (СУБД). СУБД (например, Access) представляет собой приложение для создания баз данных, как совокупности двумерных таблиц.

Приложение — программа или группа программ, предназначенных для выполнения стандартных работ. К таким приложениям относятся текстовые (например, Word), графические (CorelDraw) редакторы, электронные таблицы (Excel).

Из-за достаточной универсальности алгоритмов работы БД их стали называть алгоритмами приложений.

В силу большого разнообразия баз данных следует привести их классификацию. Для лучшего ее понимания введем необходимые основные термины. Остальные определения даны в последующих разделах.

1.3. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Как отмечалось, БД в простейшем случае представляется в виде системы двумерных таблиц. Таблицы могут быть представлены в ПК либо в виде отдельных файлов, либо размещаться в одном файле.

Файл — информация, хранимая на электронном носителе после завершения отдельных заданий и рассматриваемая в процессе обра-

ботки как единое целое. Файл имеет имя и требует некоторого объема памяти носителя, в качестве которого может выступать дискета, винчестер, компакт-диск (CD).

Поле — столбец файлового документа (таблицы). Имя поля часто называют **атрибутом**.

Домен — совокупность значений одного поля.

Универсум — совокупность значений всех полей.

Запись — строка документа. Следует отметить, что это понятие неоднозначно. В реляционной модели данных запись — строка таблицы, в сетевой модели данных — элемент структуры, аналогичный примерно таблице в реляционной модели данных.

Запись логическая — поименованная совокупность данных, рассматриваемая пользователем как одно целое.

Запись физическая (совокупность данных записываемых/считываемых одним блоком) характеризует расположение данных в физической памяти ПК.

Ключ — поле с уникальными (неповторяющимися) записями, используемое для определения места расположения записи. Ключ может состоять из совокупности полей (составной ключ), называемых **суперключом**.

Выделенный ключ — ключ, явно перечисленный вместе с реляционной схемой. В противном случае говорят о **неявном ключе**. Вводят и такие понятия как возможный ключ (ключ-кандидат), если любой из нескольких наборов полей может быть принят за составной ключ. Один из выделенных ключей называют первичным. При работе с несколькими связанными таблицами говорят о родительском ключе главной таблицы и внешнем ключе подчиненной таблицы. Иногда ключ называют **идентификатором** — атрибутом, значения которого однозначно определяют экземпляры объекта предметной области.

Предметная область — отражение в БД совокупности и объектов реального мира с их связями, относящимися к некоторой области знаний и имеющих практическую ценность для пользователя. Понятие «идентификатор» используется и в физической базе данных.

Указатель — идентификатор, который ведет к заданной записи из какой-то другой записи в физической базе данных. Здесь запись — некоторый блок данных в памяти компьютера.

Приведем перечень используемых в дальнейшем терминов, детальное пояснение которых проводится в последующих разделах данной работы.

Администратор базы данных (АБД) — лицо, отвечающее за выработку требований к БД, ее проектирование, реализацию, эффективное использование и сопровождение.

Архитектура — разновидность (обобщение) структуры, в которой какой-либо элемент может быть заменен на другой элемент, характеристики входов и выходов которого идентичны первому элементу. Понятие «принцип открытой архитектуры» используется при построении компьютера. Этот принцип означает, что вместо принтера одной марки (например, Epson) к компьютеру может быть подключен принтер другого типа (например, Hewlett Packard).

Безопасность — защита от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения.

Блокировка — неделимая операция, которая позволяет только одному процессу иметь доступ к совместно используемому ресурсу.

Вид (View) — таблица, вычисленная с помощью навигационной операции на основе исходной таблицы (таблиц). Вид может использоваться почти по тем же правилам, что и исходная таблица.

Внешняя схема — описание данных на концептуальном уровне. Как отмечалось, в реляционной БД порядок расположения полей (столбцов) таблицы безразличен. Однако для реализации следует выбрать вполне определенный порядок (схему). Чаще всего ключевые поля располагают в начале схемы.

Внутренняя схема — описание данных на физическом уровне.

Время доступа — промежуток времени между выдачей команды записи (считывания) и фактическим получением данных.

Время отклика — промежуток времени от момента запроса к БД до фактического получения данных.

Даталогическая модель — модель логического уровня, представляющая собой отображение логических связей безотносительно к их содержанию и среде хранения.

Доступ — операция поиска, чтения данных или записи их.

Задание (работа) — программа или совокупность программ и преобразуемые этими программами данные.

Защита данных — противостояние базы данных несанкционированному доступу, преднамеренному искажению или разрушению информации.

Индекс — совокупность указателей, содержащих информацию о местоположении записи. Для ускорения поиска полям сопоставляют уникальный набор (числовой или символьный). Индекс может быть представлен и несколькими полями. Если при построении БД заданы индексы, то для поиска сначала их и используют. Если индексов нет, то может проводиться длительный поиск путем перебора данных.

Концептуальный — определение, относящееся к обобщенному представлению данных, независимому от СУБД. При проектирова-

нии БД выделяют концептуальную, логическую и физическую базы данных (модели), определение которых приведено позднее.

Кортеж — совокупность полей или запись (строка).

КОДАСИЛ (CODASIL) — набор стандартов для сетевых баз данных.

Логический — определение, относящееся к представлению или описанию данных, не зависящему от запоминающей среды или вычислительной системы, однако «привязанное» к выбранной СУБД.

Машина баз данных (МБД) — вспомогательный периферийный процессор, выполняющий функции СУБД.

Метаданные — данные о данных, описание информационных ресурсов, их характеристик, местонахождения, способов использования и т. д. Например, перечень таблиц с характеристиками каждой из них (имя, объем памяти и другие параметры).

Многозначная зависимость (MV-зависимость, зависимость 1:M) — для подсхем X, Y, Z, принадлежащих схеме R, Z = R – (XY) и кортежей $t_2(X) = t_1(X)$ и $t_3(Y) = t_1(Y)$ справедливо $t_3(Z) = t_1(Z)$ и $t_3(Z) = t_2(Z)$.

Модель данных — средство абстракции, позволяющее видеть информационное содержание (обобщенную структуру), а не их конкретные значения. Выделяют, как отмечалось, иерархическую, сетевую, реляционную, объектно-ориентированную, объектно-реляционную и многомерную модели данных.

Навигация — операция, результат которой представлен единым объектом, полученным при прохождении пути по логической структуре БД. Иными словами, операция получения новой таблицы из полей связанных таблиц.

Независимость данных — возможность изменения логической и физической структуры БД без изменения представлений пользователя.

Объект — термин, обозначающий факт, лицо, событие, предмет, о котором могут быть собраны данные. В реляционных СУБД выделяют такие основные объекты, как таблицы, формы, запросы, отчеты, макросы, модули.

Объектно-ориентированное программирование — методология программирования, основанная на представлении программ в виде связанной совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию по наследованию.

Объектно-ориентированное проектирование — методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логических и физических, а также статических и динамических моделей проектируемой системы.

Отношение r на множествах (доменах) S_1, ..., S_n — подмножество декартова произведения S_1& ... &S_n. Понятие «отношение» является основным в реляционных БД. Пусть имеется таблица с двумя полями S_1 и S_2 по два значения в каждом (S_1 = {a1, a2} и S_2 = {b1, b2}, т. е. в каждом домене по два значения). «Полная» таблица имеет четыре возможных записи (a1, b1; a1, b2; a2, b1; a2, b2), которые и образуют декартово произведение. Отношением является и часть этой таблицы (например, a1, b1; a2, b1). Отношение может быть и составным: r = (r_1, ..., r_n), составленным, например, из нескольких связанных таблиц.

Подсхема — описание логического представления пользователя данной группы. Иными словами, это схема отдельного пользователя БД, если их несколько. Из подсхем может быть составлена схема БД (для всех пользователей). Нетрудно видеть, что при наличии одного пользователя подсхема является схемой.

Программа — полное и точное описание алгоритма на некотором формальном языке программирования.

Процедура — некоторая подпрограмма.

Распределенная база данных (РБД) — единая БД, представленная в виде отдельных (возможно, избыточных и перекрывающихся) разделов на разных вычислительных средствах.

Связь — ассоциация между экземплярами примитивных или агрегированных объектов (записей) данных.

Семантика — часть языка, касающаяся указания смысла и действия текста, составленного в соответствии с синтаксическими правилами. Действия текста относится к операторам на некотором языке программирования.

Синтаксис — правила, определяющие разрешенные языковые конструкции, а также последовательности расположения символов в программе.

Система баз данных — совокупность СУБД, прикладного программного обеспечения, базы данных, операционной системы и технических средств, обеспечивающих информационное обслуживание пользователей.

Система управления базой данных (СУБД) — совокупность программных средств, обеспечивающих управление БД на всех уровнях.

Системный журнал — журнал регистрации всех изменений БД.

Словарь данных — набор обобщенных описаний данных БД, обеспечивает логически централизованное хранение метаданных.

Спецификация — операция, результатом которой является новая структура, построенная на основе структур базы данных.

Структура — совокупность элементов и их связей.

Сущность — примитивный объект данных, отображающий элемент предметной области (человек, место, вещь и т. д.).

Схема данных — описание логической структуры данных, специфицированное на языке описания данных и обрабатываемое СУБД. Дело в том, что в общем случае поля таблицы (отношения) могут располагаться в произвольном порядке (семейство отношений). Для конкретного пользователя и в конкретной БД должен быть выбран и зафиксирован только один вариант порядка. Этот вариант называют схемой (пользователя).

Транзакция — процесс изменения файла или БД, вызванный передачей одного входного сообщения. Это сообщение (команду) часто тоже называют транзакцией.

Функциональная зависимость (F-зависимость, зависимость 1:1): схема Y функционально зависит от X, если для кортежей $t_1(X) = t_2(X)$, справедливо $t_1(Y) = t_2(Y)$, причем схемы X и Y могут принадлежать схеме R.

Хранимая запись — совокупность связанных элементов данных, соответствующая одной или нескольким логическим записям и содержащая все необходимые служебные данные.

Хранилище данных — предметно-ориентированный, интегрированный, привязанный ко времени и неизменный набор данных, предназначенный для поддержки принятия решений.

Целостность данных — устойчивость хранимых данных к разрушению (уничтожению), связанному с неисправностями технических средств, системными ошибками и ошибочными действиями пользователей.

Элемент данных — наименьшая единица данных, имеющая смысл при описании информации; наименьшая единица поименованных данных.

Экземпляр — отдельный экземпляр объекта, записи, элемента данных.

Язык базы данных — общий термин, относящийся к классу языков, которые используются для определения и обращения к базам данных.

Язык манипулирования данными (ЯМД) — командный язык, обеспечивающий доступ к содержимому БД и его обработку. Обработка предполагает вставку, удаление и изменение данных (операции обновления).

Язык описания данных (ЯОД) — предназначен для описания данных на концептуальном, логическом и физическом уровнях на основе соответствующих схем. Речь идет о командах по формирова-

нию структуры (шапки) таблиц и связей между ними. Эти операции могут быть обеспечены визуальным языком программирования QBE или директивным языком программирования SQL.

Язык запросов — высокоуровневый язык манипулирования данными, обеспечивающий взаимодействие пользователей с БД. Язык запросов предполагает выборку данных.

Следует отметить, что три группы операций с БД (описание, манипулирование, запрос) совмещены в языке SQL, а в некоторых СУБД — и в языке QBE.

Исходным элементом базы данных является таблица, структурные составляющие которой — поле и запись. Можно выделить две разновидности структуры таблиц: линейную и нелинейную. В линейной структуре поля располагаются последовательно друг за другом в произвольном порядке (табл. 1.11). В силу произвольности порядка для данной, конкретной реализации следует закрепить определенный вариант, называемый **схемой пользователя**.

Таблица 1.11

Таблица данных о кафедре

N_преподавателя	Фамилия	Кафедра	Дисциплина	Число_часов
115	Козлов А.И.	Информатика	Базы данных	64
...	...	...	...	...
59	Серов О.В.	Информатика	Базы данных	64

В нелинейной структуре выделяется понятие «агрегат», являющийся как бы таблицей в таблице (табл. 1.12). Агрегат может быть двух видов: вектор и повторяющиеся поля. Возможности реализации структур таблиц зависят от выбранной модели данных (МД). Реляционная и иерархическая модели данных реализуют только линейную структуру, тогда как сетевая и объектно-ориентированная модели позволяют использовать и нелинейную структуру. Особенности конкретной реализации определяются классами БД и СУБД.

Таблица 1.12

Таблица данных о студентах

Шифр студента			Оценки по семестрам			
№	Фамилия	Дисциплина	1	2	3	4
57	Скоков П.Н.	Информатика	4	5	5	4
...	...	...	...	...	...	...
12	Петров А.М.	Математика	3	5	5	4

1.4. КЛАССИФИКАЦИЯ БД И СУБД

Под классификацией понимается разделение множества на подмножества по неформально предложенному признаку. В силу многогранности баз данных и СУБД (комплекса технических и программных средств для хранения, поиска, защиты и использования данных) имеется множество классификационных признаков. Классификация БД по основным из них [28] приведена на рис. 1.12.

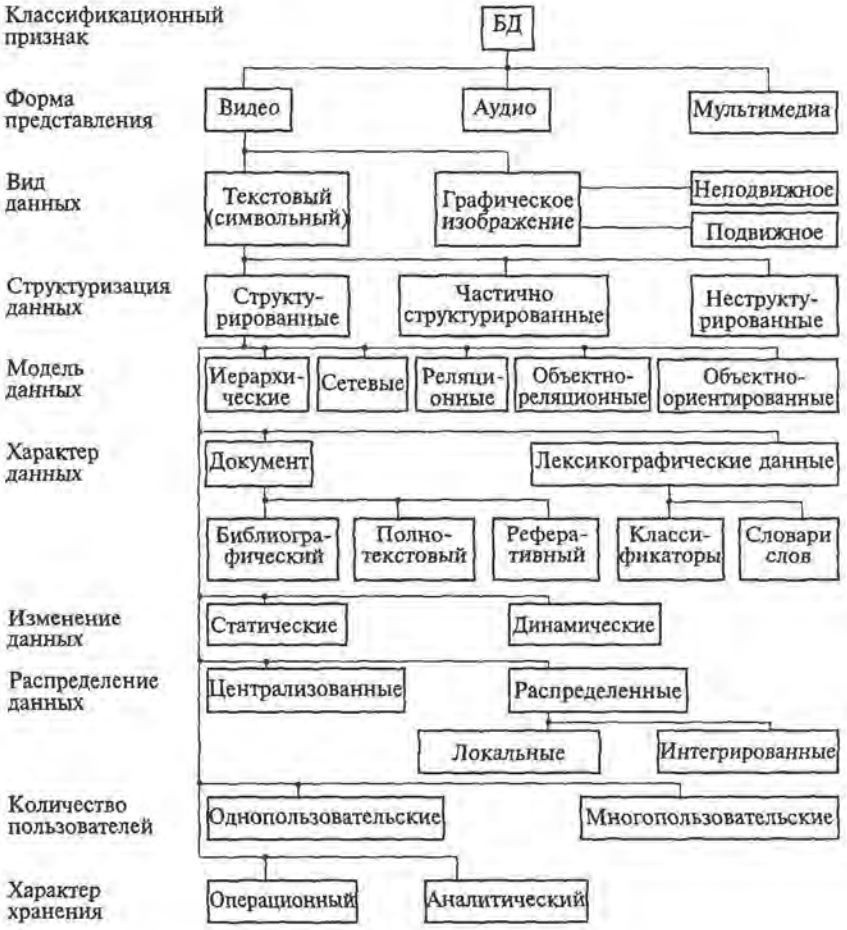


Рис. 1.12. Классификация баз данных

Особо поговорим о статических и динамических БД.

В **статических БД** частота обновления данных много ниже частоты их считывания. Данные напрямую не связаны со временем. Например, анкетные данные, которые используются гораздо чаще, чем изменяются. Именно для статических БД строилась изначально теория баз данных.

В статических БД данные чаще изменяются, чем добавляются. В связи с этим основным требованием к ним стало простота обновления, что достигается разделением таблицы на несколько в процессе *нормализации*. Формируются чаще всего первая, вторая и третья нормальная формы, реже — четвертая и пятая. Использование второй и выше форм говорит о глубокой нормализации.

В последнее время все чаще обращаются к **динамическим БД**, в которых частоты считывания и обновления данных соизмеримы. В динамических БД время выступает явно в виде понятий момента времени (дата) или интервала времени (семестр, месяц, год). Например, данные об успеваемости студентов групп за время обучения (по семестрам).

Для таких БД более характерно не изменение, а добавление данных и процедура глубокой нормализации теряет актуальность. Нормализация может ограничиваться первой нормальной формой.

Отдельно следует классифицировать системы управления базами данных (рис. 1.13). Базы данных могут классифицироваться и с точки зрения экономической [8]: — бесплатные и платные (бесприбыльные, коммерческие); по форме собственности (государственные, негосударственные); по степени доступности (общедоступные, с ограниченным кругом пользователей).

В настоящее время имеются дополнительные, необщепринятые предложения [3] по выделению таких классов, как активные БД

Рис. 1.13. Классификация СУБД

(фактически — экспертные системы, ранее называвшиеся дедуктивными базами данных); пространственные БД (связанные с хранением графических файлов, например, географических карт); временные БД, в которых время присутствует в явном виде в качестве полей таблиц. Поскольку два последних класса входят в состав других классов и самостоятельного значения пока не имеют, от этой линии классификации здесь воздержимся.

В дальнейшем сосредоточим внимание на видеобазах данных-текстов (таблиц) со структурированными данными документального характера. Речь пойдет об открытых СУБД, как правило, операционного типа.

До середины 90-х годов XX в. под базой данных понимали статические БД, которые впоследствии получили название операционных (транзакционных) БД, а за рубежом — OnLine Transaction Processing (OLTP).

К середине 90-х годов в базах данных класса OLTP скопилось столько хронологической информации, что объем БД резко возрос, а быстродействие начало падать. Например, в работе деканата чаще всего требуются детальные данные о текущем учебном годе. В то же время в БД хранятся ретроспективные данные и за предыдущие годы. Такие данные необходимы значительно реже и чаще всего в агрегированном виде. Например, выдать фамилии студентов, которые три последних семестра получали только отличные оценки.

Стало ясно, что ретроспективную информацию следует периодически передавать в отдельную БД. К тому же выяснилось, что ретроспективные данные обладают новым качеством: они позволяют вырабатывать стратегические решения. Возникла возможность формирования систем поддержки принятия стратегических решений (СППР). Такие системы получили за рубежом название OnLine Analytic Processing (OLAP).

Связь OLTP и OLAP показана на рис. 1.14. OLAP служит как бы дополнением OLTP.

OLAP сразу была оценена по достоинству менеджерами, поскольку позволяла: существенно повысить эффективность труда руководителей различного ранга; повысить конкурентоспособность фирм; получать дополнительно высокую прибыль.

Приведенные характеристики системы OLAP позволяют заметить, что при проектировании этой системы, как и при проектировании OLTP, возникают сложности, определяемые следующими обстоятельствами:

• потребностью в значительных вычислительных ресурсах для обработки большого объема данных;

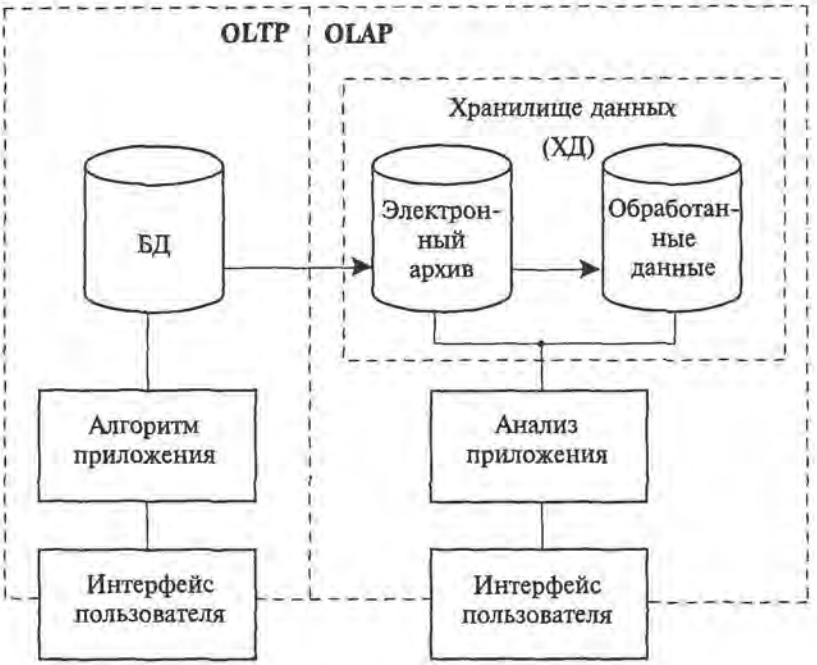


Рис. 1.14. Соотношение OLTP и OLAP

• проблемами, связанными с незаполненностью (отсутствием данных) или отсутствием необходимых полей в операционной БД;
• увеличением сложности сопровождения и реструктуризации;
• значительной длительностью создания и заполнения OLAP.

OLAP, по сравнению с OLTP характеризуется несколько другими свойствами, как это видно из табл. 1.13.

В дальнейших рассуждениях, при отсутствии специальной оговорки, под СУБД будем понимать операционную базу данных.

Таблица 1.13

Свойства данных в OLTP и OLAP

Свойство	OLTP	OLAP
Назначение данных	Оперативный поиск, несложная обработка	Аналитическая обработка: прогнозирование, моделирование, анализ и выявление связей, выявление статистических закономерностей
Уровень агрегации данных	Детальные данные	Агрегированные данные

Продолжение табл. 1.13

Свойство	OLTP	OLAP
Период хранения данных	До года	До нескольких десятков лет
Изменчивость данных	Изменяются	Добавляются
Упорядочение данных	По любому полю	По хронологии
Объем обрабатываемой информации	Небольшой	Очень большой
Скорость обработки	Средняя	Очень высокая
Критерий эффективности работы	Количество транзакций в единицу времени	Скорость выполнения сложных запросов
Загрузка	Часто и небольшими порциями	Редко и очень большими порциями

1.5. СОСТАВ СУБД И РАБОТА БД

СУБД представляет собой оболочку, с помощью которой после построения структуры таблиц, задания связей между таблицами и заполнения таблиц данными получается соответствующая база данных. В связи с этим полезно поговорить о системе программно-технических, организационных и «человеческих» составляющих (рис. 1.15).

Программные средства включают трансляторы и систему управления, обеспечивающую ввод—вывод, обработку и хранение инфор-

Рис. 1.15. Состав СУБД

мации, создание, модификацию и тестирование БД. Базовыми внутренними языками программирования являются языки четвертого поколения. В качестве базовых языков могут использоваться C, C++, Pascal, Object Pascal. Язык C++ позволяет строить программы как на языке Visual Basic с его широким спектром возможностей, более близкий и понятный даже пользователю-непрофессионалу, так и на непроцедурном (декларативном) языке структурированных запросов SQL. Ранее отмечалось, что исторически для системы управления базой данных сложились три языка:

1) язык описания данных (ЯОД), называемый также языком описания схем — для построения структуры («шапки») таблиц БД;

2) язык манипулирования данными (ЯМД) — для заполнения БД данными и операций обновления (запись, удаление, модификация);

3) язык запросов — язык поиска наборов величин в файле в соответствии с заданной совокупностью критериев поиска и выдачи затребованных данных без изменения содержимого файлов и БД (язык преобразования критериев в систему команд).

В настоящее время функции всех трех языков выполняет язык SQL, относящийся к классу языков, базирующихся на исчислении кортежей.

Вместе с тем сохранились и языки запросов, например, язык запросов по примеру Query By Example (QBE) класса реляционного исчисления доменов. Отметим, что эти языки в качестве «информационной единицы» БД используют отдельную запись. С помощью языков БД создаются приложения, базы данных и интерфейс пользователя, включающий экранные формы, меню, отчеты. При создании БД на базе СУБД Paradox эти элементы (объекты) фиксируются в отдельных файлах. В СУБД Access, Interbase все созданные объекты размещаются в одном файле.

Для работы с созданной БД пользователю или администратору БД следует иметь перечень файлов — таблиц с описанием состава их данных (структуры, схемы). Для этого создается специальный файл, называемый *словарем данных* (репозитарием, словарем-справочником, энциклопедией). Описание БД относится к метаинформации.

В качестве технических средств могут выступать супер- или персональные компьютеры с соответствующими периферийными устройствами.

Организационно-методические средства — это совокупность инструкций, методических и регламентирующих материалов, описаний структуры и процедуры работы пользователя с СУБД и БД.

Пользователей возможно разделить на две основные категории:

- конечные пользователи (КП);
- администраторы баз данных (АБД).

Особо следует поговорить об администраторе базы данных. Естественно, что база данных строится для конечного пользователя, однако первоначально предполагалось, что КП не смогут работать без специалиста-программиста, которого назвали администратором базы данных. С появлением СУБД они взяли на себя значительную часть функций АБД, особенно для БД с небольшим объемом данных. Однако для крупных централизованных и распределенных баз данных потребность в АБД сохранилась. В широком плане под АБД понимают системных аналитиков, проектировщиков структур данных и информационного обеспечения, проектировщиков технологии процессов обработки, системных и прикладных программистов, операторов, специалистов в предметной области и по техническому обслуживанию. Иными словами — в крупных базах данных это могут быть коллективы специалистов.

В обязанности АБД входит: анализ предметной области, статус информации и пользователей; проектирование структуры и модификация данных; задание и обеспечение целостности; загрузка и ведение БД; защита данных; обеспечение восстановления БД; сбор и статистическая обработка обращений к БД, анализ эффективности функционирования БД; работа с пользователем.

Одним из важнейших инструментов АБД является словарь.

В заключение отметим, что в работе с базами данных выделяют следующие процедуры:

- построение (создание, проектирование) БД;
- использование БД;
- функционирование БД.

Эти процедуры определяют содержание (составляющие) теории БД: *создание, использование, функционирование.*

При реализации БД основными элементами структуры БД являются собственно база данных (система таблиц с данными); интерфейс пользователя, алгоритм приложения (при современном подходе к проектированию БД или алгоритм преобразования — при традиционном подходе).

Перейдем к более подробному рассмотрению OLAP.

В составе OLAP можно выделить такие [2] архитектурные элементы (рис. 1.16):

- хранилище данных (ХД);
- менеджер загрузки;

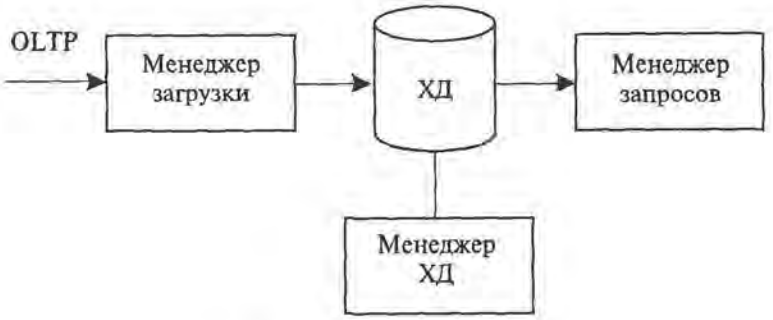


Рис. 1.16. Состав OLAP

- менеджер хранилища данных;
- менеджер запросов.

Хранилище данных, в первом приближении, возможно (по аналогии с OLTP) считать базой данных, тогда как систему OLAP—СУБД. В нем можно условно выделить электронный архив, хранящий детальные ретроспективные данные, и агрегированные (обработанные) данные.

ХД реализуется с помощью многомерной модели, которая имеет несколько разновидностей [17]: собственно многомерная (Multidimensional OLAP—MOLAP), реляционная (Relational OLAP—ROLAP), гибридная (Hibrid OLAP—HOLAP). Более подробно эти разновидности рассмотрены в гл. 8.

Менеджер загрузки осуществляет преобразование данных, поступающих из операционных БД, и прежде всего — форматирование по «стандарту» OLAP.

Менеджер хранилища данных выполняет следующие операции:

- анализ непротиворечивости исходных данных;
- создание необходимых индексов и видов;
- денормализацию;
- резервное копирование.

Менеджер запросов управляет пользовательскими запросами, возможно с графиками процесса выполнения запроса.

Контрольные вопросы

1. Что такое данные, информация, знания?
2. Дайте определение базы данных (БД).
3. Каково назначение БД?
4. Дайте определение понятиям «файл», «запись», «атрибут», «домен», «поле», «ключ», «суперключ», «архитектура», «схема данных», «модель данных», «кортеж», «словарь данных».

5. Дайте определения понятиям «предметная область», «приложение», «программа», ЯОД, ЯМД.
6. Дайте классификацию СУБД и БД.
7. Охарактеризуйте состав СУБД.
8. Покажите соотношение СУБД и АБД.
9. Перечислите процедуры работы БД.
10. Назовите составляющие теории баз данных.
11. Перечислите основные элементы структуры БД с позиций ее реализации.
12. Каково назначение OLTP и OLAP? соотношение их свойств?
13. Опишите состав OLAP.
14. Назовите разновидности многомерной модели.

Глава 2

Концепция баз данных

Концепция в общем смысле представляет некоторую систему взглядов на процесс или явление. Составными частями концепции являются совокупность принципов и методология. Под методологией понимается совокупность методов решения проблемы.

Принцип — правила, которыми следует руководствоваться в деятельности. Часто принципы формулируются в виде ограничений и требований, в частности, требований к базам данных.

2.1. ТРЕБОВАНИЯ, ПРЕДЪЯВЛЯЕМЫЕ К БАЗАМ ДАННЫХ

С современных позиций следует порознь рассматривать требования, предъявляемые к транзакционным (операционным) базам данных и к хранилищам данных.

Первоначально перечислим основные требования, которые предъявляются к операционным базам данных, а следовательно, и к СУБД, на которых они строятся.

1. Простота обновления данных. Под операцией обновления понимают добавления, удаления и изменения данных.

2. Высокое быстродействие (малое время отклика на запрос). Время отклика — промежуток времени от момента запроса к БД и фактическим получением данных. Похожим является термин время доступа — промежуток времени между выдачей команды записи (считывания) и фактическим получением данных. Под доступом понимается операция поиска, чтения данных или записи их.

3. Независимость данных.

4. Совместное использование данных многими пользователями.

5. Безопасность данных — защита данных от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения.

6. Стандартизация построения и эксплуатации БД (фактически СУБД).

7. Адекватность отображения данных соответствующей предметной области.

8. Дружелюбный интерфейс пользователя.

Важнейшими являются первые два противоречивых требования: повышение быстродействия требует упрощения структуры БД, что, в свою очередь, затрудняет процедуру обновления данных, увеличивает их избыточность (см. гл. 4).

Независимость данных — возможность изменения логической и физической структуры БД без изменения представлений пользователей. Независимость данных предполагает инвариантность к характеру хранения данных, программному обеспечению и техническим средствам. Она обеспечивает минимальные изменения структуры БД при изменениях стратегии доступа к данным и структуры самих исходных данных. Это достигается, как будет показано далее, «смещением» всех изменений на этапы концептуального и логического проектирования с минимальными изменениями на этапе физического проектирования [2].

Безопасность данных включает их целостность и защиту. Целостность данных — устойчивость хранимых данных к разрушению и уничтожению, связанных с неисправностями технических средств, системными ошибками и ошибочными действиями пользователей.

Она предполагает:

- отсутствие неточно введенных данных или двух одинаковых записей об одном и том же факте;
- защиту от ошибок при обновлении БД;
- невозможность удаления порознь (каскадное удаление) связанных данных разных таблиц;
- неискажение данных при работе в многопользовательском режиме и в распределенных базах данных;
- сохранность данных при сбоях техники (восстановление данных).

Целостность обеспечивается триггерами целостности — специальными приложениями-программами, работающими при определенных условиях. Для некоторых СУБД (например, Access, Paradox) триггеры являются встроенными.

Защита данных от несанкционированного доступа предполагает ограничение доступа к конфиденциальным данным и может достигаться:

- введением системы паролей;
- получением разрешений от администратора базы данных (АБД);
- запретом от АБД на доступ к данным;
- формированием видов — таблиц, производных от исходных и предназначенных конкретным пользователям.

Три последние процедуры легко выполняются в рамках языка структурированных запросов Structured Query Language — SQL, часто называемом SQL2.

Стандартизация обеспечивает преемственность поколений СУБД, упрощает взаимодействие БД одного поколения СУБД с одинаковыми и различными моделями данных. Стандартизация (ANSI/SPARC) осуществлена в значительной степени в части интерфейса пользователя СУБД и языка SQL. Это позволило успешно решить задачу взаимодействия различных реляционных СУБД как с помощью языка SQL, так и с применением приложения Open DataBase Connection (ODBC). При этом может быть осуществлен как локальный, так и удаленный доступ к данным (технология клиент—сервер или сетевой вариант).

Перейдем к требованиям, предъявляемым к хранилищам данных, которые структурно являются продолжением операционных баз данных.

Пусть в базе данных имеются данные об успеваемости студентов третьего курса, при этом текущими являются пятый и шестой семестры. Данные за первые четыре семестра находятся (переданы) в хранилище данных (ХД), т. е. фактически в дополнительной, специфической базе данных. Необходимо запросить в хранилище фамилии студентов, которые первые четыре семестра учились только на отлично.

Иными словами, данные из операционной БД периодически передаются в электронный архив (в рассмотренном примере — данные за первые четыре семестра), а затем могут быть обработаны в соответствии с запросом пользователя.

Поскольку данные в хранилище практически не изменяются, а лишь добавляются, требование простоты обновления становится неактуальным. На первое место — в силу значительного объема данных в хранилище — выходит требование высокого быстродействия.

К хранилищам данных предъявляются следующие дополнительные требования [2]:

- высокая производительность загрузки данных из операционных БД;
- возможность фильтрования, переформатирования, проверки целостности исходных данных, индексирования данных, обновления метаданных;
- повышенные требования к качеству исходных данных в части обеспечения их непротиворечивости, поскольку они могут быть получены из разных источников;
- высокая производительность запросов;

- обеспечение высокой размерности;
- одновременность доступа к ХД;
- наличие средств администрирования.

Поддержка анализа данных соответствующими методами (инструментами).

Как отмечено в [2] Э.Ф. Кодд на основе своего опыта предъявил следующие требования к системе OLAP.

1. Многомерное концептуальное представление данных.
2. Прозрачность технологии и источников данных.
3. Доступность к источникам данных при использовании различных моделей данных.
4. Неизменная производительность подготовки отчетов при росте объема, количества измерений, процедур обобщения данных.
5. Использование гибкой, адаптивной, масштабируемой архитектуры клиент—сервер.
6. Универсальность измерений (формулы и средства создания отчетов не должны быть привязаны к конкретным видам размерностей).
7. Динамическое управление разреженностью матриц (пустые значения NULL должны храниться эффективным образом).
8. Многопользовательская поддержка.
9. Неограниченные операционные связи между размерностями.
10. Поддержка интуитивно понятных манипуляций с данными.
11. Гибкость средств формирования отчетов.
12. Неограниченное число измерений и уровней обобщения.

Перечисленные требования отличны от требований к операционным БД, что вызвало появление специализированных БД — хранилищ данных.

2.2. КОНЦЕПЦИЯ ПОСТРОЕНИЯ БД

Концепция предполагает изложение основных положений чего-либо. Описание концепции БД невозможно выполнить без учета хронологии [6].

1. Инженерные и экономические задачи. Первоначально (начало 60-х г. XX в.) использовалась файловая система хранения. Для решения преимущественно инженерных задач, характеризующихся небольшим количеством данных и значительным объемом вычислений, данные хранились непосредственно в программе. Применялся последовательный способ организации данных, имелась их высокая избыточность, идентичность логической и физической структур и полная зависимость данных.

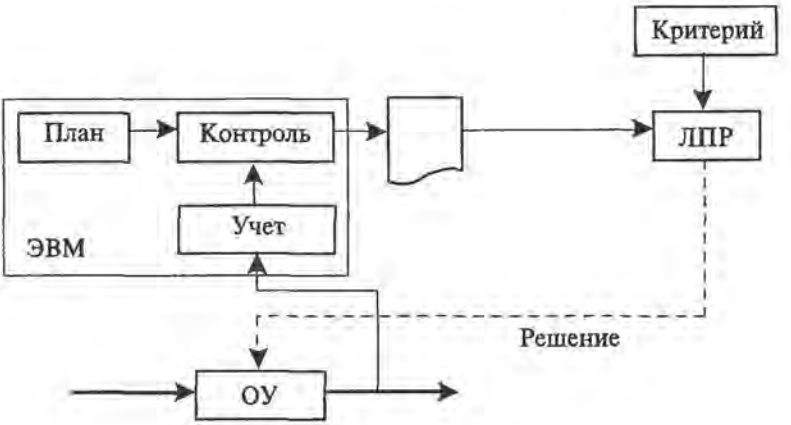


Рис. 2.1. Информационно-поисковая система

С появлением экономико-управленческих задач (информационная система руководства — MIS), отличающихся большими объемами данных и малой долей вычислений, указанная организация данных оказалась неэффективной. Требовалось упорядочение данных, которое, как выяснилось, возможно было проводить по двум критериям: использование (информационные массивы); хранение (базы данных).

2. Информационно-поисковые и информационно-советующие системы управления. Следует отметить [43], что экономические задачи часто связаны с управлением организационными системами. По характеру применения компьютеров такие системы возможно разделить на информационно-поисковые (рис. 2.1), получившие также название «традиционные», и информационно-советующие или современные (рис. 2.2) системы. Сначала шло построение и изучение традиционных систем.

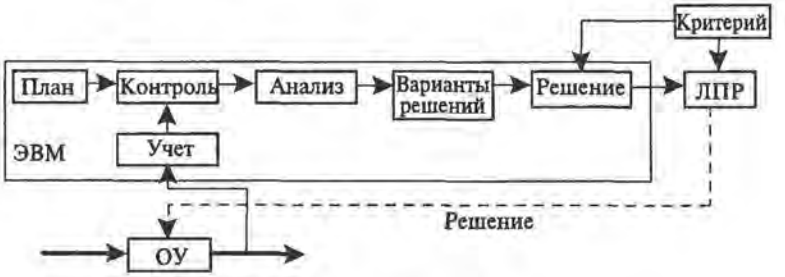


Рис. 2.2. Информационно-советующая система

3. Информационные массивы и базы данных. Первоначально в информационно-поисковых системах применяли информационные массивы. При этом возникала необходимость хранения избыточной информации при дефиците компьютерной памяти. Выяснилось также, что алгоритмы задач более подвижны, чем данные для них. При довольно частом изменении алгоритмов в процессе совершенствования систем управления каждый раз требовалось проводить трудоемкую процедуру создания новых массивов. В этих условиях стало ясно превосходство баз данных, несмотря на их более сложную структуру по сравнению с системой массивов. В дальнейшем базы данных стали снабжаться программной составляющей, позволяющей легко реализовать и оперативно изменять алгоритмы приложения.

4. Модели данных. Использование файлов для хранения только данных (рис. 2.3, *а*) предложено МакГри в 1959 г. Были разработаны методы доступа (в том числе — произвольного) к таким файлам, при этом физическая и логическая структуры уже различались, а физическое расположение данных можно было менять без изменения логического представления.

В 1963 г. С. Бахман построил первую промышленную базу данных IDS с сетевой моделью данных, которая все еще характеризовалась избыточностью данных и ее использованием только для одного приложения. Доступ к данным осуществлялся с помощью соответствующего программного обеспечения.

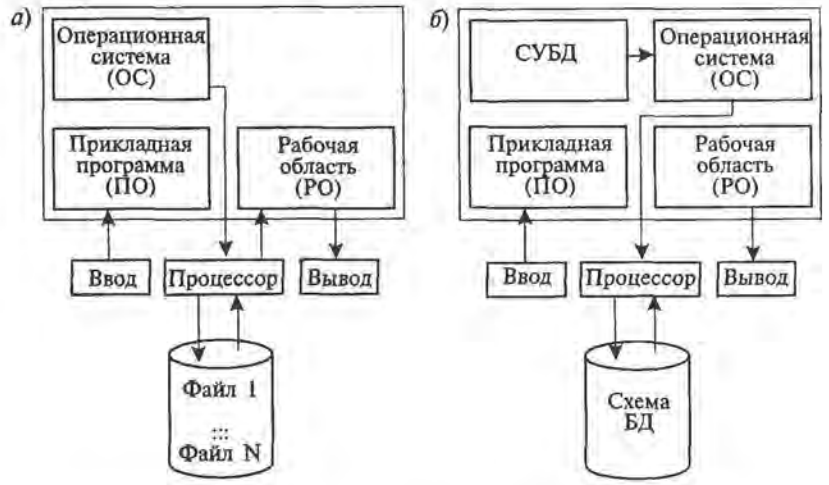


Рис. 2.3. Файловая система (*а*) и СУБД (*б*) для хранения данных

В 1969 г. сформировалась группа, создавшая набор стандартов CODASYL (КОДАСИЛ) для сетевой модели данных. Фактически начала тогда использоваться (рис. 2.3, *б*) современная архитектура базы данных. Существенный скачок в развитии технологии баз данных произошел в 1970 г., когда М. Кодд предложил парадигму реляционной модели данных. Под **парадигмой** понимается научная теория, воплощенная в систему понятий, отражающих существенные черты действительности. Теперь логические структуры могли быть получены из одних и тех же физических данных, т. е. доступ к одним и тем же физическим данным мог осуществляться различными приложениями по разным путям. Стало возможным обеспечение целостности и независимости данных.

В конце 70-х годов XX в. появились современные СУБД, обеспечивающие физическую и логическую независимость, безопасность данных, обладающие развитыми языками БД.

В начале 90-х годов реляционные БД получили наиболее широкое распространение, особенно при использовании персональных компьютеров. Появились разнообразные СУБД, рассчитанные как на пользователя-профессионала (в программировании), так и на пользователя-непрофессионала, предназначенные для построения и небольших (по объему памяти), и сверхбольших БД, работающие как в локальном, так и в сетевом режимах. При этом базы данных строились как статические (в зарубежной терминологии — операционные, транзакционные, OnLine Transactional Processing — OLTP).

К середине 90-х годов в базах данных накопилось такое количество информации, что ее стало возможным использовать для аналитических процедур выработки решений-советов. Появились динамические (аналитические) базы данных, называемые за рубежом OnLine Analytical Processing — OLAP. Их основными составляющими стали электронный архив и хранилище данных (Data Warehouse).

Одновременно выявились недостатки реляционных БД, у которых появились конкуренты в виде объектно-ориентированных баз данных.

Последнее десятилетие характеризуется появлением распределенных и объектно-ориентированных баз данных, характеристики которых определяются приложениями средств автоматизации проектирования и интеллектуализации БД.

Прежде, чем рассматривать процедуры работы с базой данных, дадим набор характеристик БД и пояснения к нему.

5. Подходы к построению БД. Они базируются на двух подходах к созданию автоматизированной системы управления (АСУ). Первый из них, широко использовавшийся в 80-е годы и потому полу-

чивший название *классического* (*традиционного*), связан с автоматизацией документооборота (совокупность документов, движущихся в процессе работы предприятия). Исходными и выходными координатами являлись документы, как это видно из примера 2.1. Трансформация входных документов в выходные осуществляется по *алгоритму преобразования*.

Пример 2.1. Задача ставится следующим образом. Имеется система ручных документов, форма одного из которых показана в табл. 2.1. Необходимо с помощью БД получить — по регламенту или по запросу — информацию в виде другой системы документов, форма одного из которых приведена в табл. 2.2.

Таблица 2.1

Студент

Номер группы	Номер зачетной книжки	Номер студенческого билета	Фамилия	Адрес
И4	И-99/12	И-99/12	Петров	Наличный пер., 5
И3	И-98/1	И-98/1	Иванов	Невский пр. 7, кв. 3
...	...	...	...	...

Таблица 2.2

Список студентов группы И4

Номер зачетной книжки	Фамилия
И-99/12	Петров
И-99/17	Сергеев
...	...

Использовался следующий тезис. Данные менее подвижны, чем алгоритмы, поэтому следует создать универсальную БД, которую затем можно использовать для любого алгоритма. Однако вскоре выяснилось, что создание универсальной БД проблематично. Господствовавшая до недавнего времени концепция интеграции данных при резком увеличении их объема оказалась несостоятельной. Более того, стали появляться приложения (например, текстовые, графические редакторы), базирующихся на широко используемых стандартных алгоритмах. Выявились стандартные алгоритмы и в управлении (бизнесе), как это следует из примера 2.2.

Пример 2.2. Используем компьютер для поддержки процедуры принятия решений менеджера в процедуре принятия специалистов на работу (комплектование кадрами исследовательской фирмы). Часть людей уже работает (в штате фирмы), необходимо провести доукомплекацию кадров. На основе анкетных данных о претендентах на вакантные места штатного расписания компьютер, в соответствии с заложенной проектировщиком системой правил, выдает менеджеру решения—советы о должностях, на которые следует принять поступающих. Окончательное решение остается за менеджером.

Если менеджер сомневается в правильности полученного компьютером решения, он может запросить объяснение в виде системы использованных при решении правил.

Если количество рекомендованных к приему превышает число вакансий в штатном расписании, менеджер может скорректировать либо правила (количественную составляющую), либо результаты их работы. Решения менеджера вводятся в компьютер.

Перечисленные процедуры имеют место на каждом из нескольких (по умолчанию — из трех) интервалов (циклов) времени.

В завершение компьютер выдает на экран итоговые результаты работы менеджера. Данный пример реализован на компьютере и подробно описан в гл. 15.

К 90-м годам XX в. сформировался второй, *современный* подход, связанный с автоматизацией управления. Он предполагает первоначальное выявление стандартных алгоритмов приложений (алгоритмов бизнеса в зарубежной терминологии), под которые определяются данные, а стало быть, и база данных. Объектно-ориентированное программирование только усилило значимость этого подхода. Состав БД для различных подходов представлен на рис. 2.4.

Рис. 2.4. Схема классического (*а*) и современного (*б*) подхода при построении БД

В работе БД возможны одно- и многопользовательский режимы. В последнем случае несколько пользователей подключаются к одному компьютеру через разные порты.

6. Восходящее и нисходящее проектирование БД. Первое применяют в распределенных БД при интеграции спроектированных локальных баз данных, которые могут быть выполнены с использованием различных моделей данных. Более характерным для централизованных БД является нисходящее проектирование.

В последующих разделах первоначально будет рассмотрен классический подход для централизованных БД, а затем — современный. Распределенным БД посвящена гл. 10—12 настоящей работы.

Работа с базами данных может быть представлена в виде схемы, приведенной на рис. 2.5. Из нее видно, что следует выделять методологию создания, методологию использования и методологию фун-

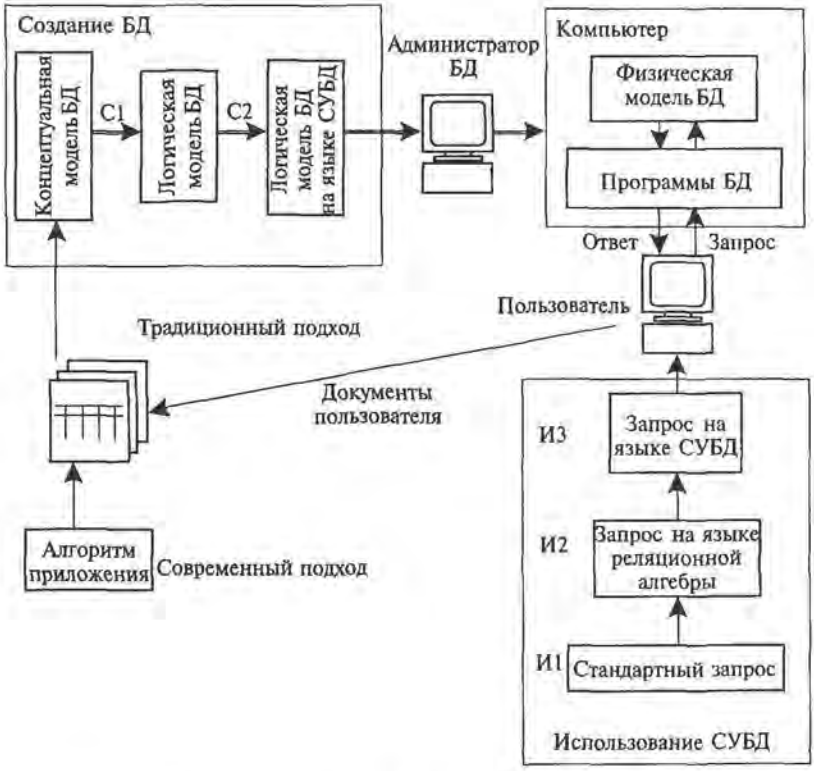


Рис. 2.5. Этапы создания (С1, С2) и использования (И1—И3) БД

кционирования БД. Методология БД определяется в процедуре проектирования, но проявляется и в процедуре использования.

7. **Хранилище данных** — предметно-ориентированный, интегрированный, привязанный ко времени и неизменный набор данных, предназначенный для поддержки принятия решений [2, 3, 17]. В соответствии с определением хранилище данных ориентировано не на алгоритм приложения, как (операционная) БД, а на предметную область.

Интегрированность определяется тем фактом, что источниками данных могут быть несколько БД, которые могут иметь разные форматы данных и степень заполнения БД. Эти данные должны быть приведены к «стандарту», используемому в ХД.

Привязка ко времени означает, что исходные данные характеризуют какой-то интервал времени, при этом время присутствует в БД явно. В силу этого вновь поступающие данные не изменяют прежние данные в ХД, а дополняют их.

Рассмотрим методологические аспекты БД и ХД.

2.3. МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Существует много разновидностей методологии рассмотрения баз данных в классическом подходе, однако чаще всего придерживаются методологии ANSI/SPARC [4, 5, 10—12], схема которой представлена на рис. 2.6. Совокупность процедур проектирования централизованной БД можно разделить на четыре этапа.

На этапе *формулирования и анализа требований* устанавливаются цели организации, определяются требования к БД. Они состоят из общих требований, определенных в разд. 2.1, и специфических требований. Для формирования специфических требований обычно используется методика интервьюирования персонала различных уровней управления. Все требования документируются в форме, доступной конечному пользователю и проектировщику БД.

Этап *концептуального проектирования* заключается в описании и синтезе информационных требований пользователей в первоначальный проект БД. Исходными данными могут быть совокупность документов пользователя (см. рис. 2.5) при классическом подходе или алгоритмы приложений (алгоритмы бизнеса) при современном подходе. Результатом этого этапа является высокоуровневое представление (в виде системы таблиц БД) информационных требований пользователей на основе различных подходов. Сначала выбирается модель БД. Затем с помощью ЯОД создается структура БД, которая

Рис. 2.6. Этапы проектирования операционных БД

затем заполняется данными с помощью команд ЯМД, систем меню, экранных форм или в режиме просмотра таблиц БД. Здесь же обеспечивается защита и целостность (в том числе — ссылочная) данных с помощью СУБД или путем построения триггеров.

В процессе *логического проектирования* высокоуровневое представление данных преобразуется в структуру используемой СУБД. Основной целью этапа является устранение избыточности данных с использованием специальных правил — нормализации (рис. 2.6). При этом минимизируется повторение данных и возможные структурные изменения БД при процедурах обновления. Это достигается разделением (декомпозицией) одной таблицы на две или более с последующим использованием при запросах операции навигации. Заметим, что навигационный поиск снижает быстродействие БД, т. е. увеличивает время отклика на запрос. Полученная логическая структура БД может быть оценена количественно с помощью различных характеристик (число обращений к логическим записям, объем данных в каждом приложении, общий объем данных). На основе этих оценок логическая структура может быть усовершенствована с целью достижения большей эффективности.

Специального рассмотрения заслуживает процедура управления БД. Она наиболее проста в однопользовательском режиме. В многопользовательском режиме и в распределенных БД процедура сильно усложняется. При одновременном доступе нескольких пользователей без принятия специальных мер возможно нарушение целостности. Для устранения этого явления используют систему транзакций и режим блокировки таблиц или отдельных записей. Особенности блокирования и варианты блокировки далее будут рассмотрены отдельно.

На этапе *физического проектирования* решаются вопросы, связанные с производительностью системы, определяются структуры хранения данных и методы доступа.

Взаимодействие между этапами проектирования и словарной системой необходимо рассматривать отдельно. Процедуры проектирования могут использоваться независимо в случае отсутствия словарной системы. Сама словарная система может рассматриваться как элемент автоматизации проектирования.

Средства проектирования и оценочные критерии используются на всех стадиях разработки. В настоящее время неопределенность при выборе критериев является наиболее слабым местом в проектировании БД. Это связано с трудностью описания и идентификации большого числа альтернативных решений.

В то же время существует много критериев оптимальности, являющихся неизмеримыми свойствами, трудно выразимыми в количественном представлении или в виде целевой функции.

К качественным критериям могут относиться гибкость, адаптивность, доступность для новых пользователей, совместимость с другими системами, возможность конвертирования в другую вычислительную среду, возможность восстановления данных, возможность распределения и расширения.

Проще обстоит дело при работе с количественными критериями, к которым относятся время ответа на запрос, стоимость модификации, стоимость памяти, время на создание, стоимость на реорганизацию. Затруднение может вызывать противоречие критериев друг другу.

Процесс проектирования является длительным и трудоемким и обычно продолжается несколько месяцев.

2.4. МЕТОДОЛОГИЯ ИСПОЛЬЗОВАНИЯ БАЗ ДАННЫХ

БД используются обычно не самостоятельно, а являются компонентой различных информационных систем: банков данных, информационно-поисковых и экспертных систем, систем автоматизированного проектирования, автоматизированных рабочих мест, автоматизированных систем управления.

В БД имеется три уровня представления данных (см. рис. 2.6): концептуальная, логическая и физическая базы данных. В процедуре использования чаще всего имеют дело с логической и значительно реже с концептуальной и физической моделями.

Словарь данных представляет собой как бы внутреннюю БД, содержащую централизованные сведения о всех типах данных, их имена, структуру, а также информацию об их использовании. Преимущество словаря данных — в эффективном накоплении и управлении информационными ресурсами предметной области. Его применение позволяет уменьшить избыточность и противоречивость данных при их вводе, осуществить простое и эффективное управление при их модификации, упростить процедуру проектирования БД за счет централизации управления данными, установить связи с другими пользователями. Таким образом, словарь данных содержит обобщенное представление всех трех уровней: концептуального, логического и физического.

В логическом представлении применяются следующие виды моделей данных: иерархические, сетевые, реляционные, объектно-ориентированные (объектно-реляционные).

Иерархическая модель служит разновидностью сетевой, являющейся совокупностью деревьев (лесом).

Сетевая модель допускает только бинарные связи «многие к одному» и использует для описания модель ориентированных графов.

Реляционная модель использует представление данных в виде таблиц (реляций, связей). В ее основе лежит математическое понятие теоретико-множественного отношения: она базируется на реляционной алгебре и теории отношений.

В *объектно-ориентированной модели* используются понятия класса, объекта, метода.

В процессе использования БД имеются операции обновления (запись, удаление, модификация данных) и запрос-ответ (чтение).

В общем случае процесс запроса состоит из ряда этапов (И1—И3 на рис. 2.5). Пользователь должен знать структуру БД или обратиться к АБД.

На этапе И1 пользователь должен выяснить, какие формы документов ему нужны. Это могут быть не только логические модели пользователя, но и различные их модификации при разных сочетаниях полей. Поскольку логические (а тем более модифицированные логические) модели могут отличаться от логической модели БД, следует определить, какие сочетания полей необходимы для выводимых машинных документов.

Эти сочетания образуются с помощью элементарных правил (этап И2), изучаемых реляционной алгеброй и реляционным исчислением. Далее правила следует трансформировать в соответствующие варианты обращения к СУБД через ее интерфейс. Это могут быть меню, экранные формы, язык программирования (например, SQL), запрос по примеру, режим просмотра таблиц БД. Результат может быть представлен в виде таблиц или отчетов.

При эксплуатации БД используют и две специфические операции: навигацию и спецификацию.

Для работы с БД используется специальный обобщенный инструментарий в виде СУБД, предназначенный для управления БД и обеспечения интерфейса пользователя. Существует два основных направления реализации СУБД: программное и аппаратное.

Программная реализация (в дальнейшем СУБД) представляет собой набор программных модулей, работает под управлением конкретной ОС и выполняет следующие функции: описание данных на концептуальном и логическом уровнях; загрузку данных; хранение данных; поиск и ответ на запрос (транзакцию); внесение изменений; обеспечение безопасности и целостности; предоставление пользователю языковых средств: языка описания дан-

ных (ЯОД), языка манипулирования данными (ЯМД), языка запросов.

Аппаратная реализация предусматривает использование и так называемых машин баз данных. Их появление вызвано возросшими объемами информации и требованиями к скорости доступа.

Таким образом, в соответствии с рис. 2.4, 2.5, теоретические вопросы можно скомпоновать в две группы (см. рис. 2.6).

1. Общая теория баз данных. Сюда относятся вопросы, не зависящие от моделей данных:

а) математический аппарат баз данных (см. гл. 3);

б) описание структуры БД, в том числе различных МД с их сравнительными характеристиками, выбор МД, структурные преобразования БД.

2. Теория реляционных БД (см. гл. 4). Для них наиболее продвинута прикладная математическая теория БД. Она включает три фактически автономные раздела (рис. 2.6):

а) организацию структур таблиц БД (прежде всего — нормирование), их заполнение и обеспечение составляющих целостности — при проектировании БД;

б) обеспечение целостности данных и их восстановление за счет соответствующих характеристик СУБД — в процессе работы СУБД;

в) организацию запросов и обновления данных — при эксплуатации БД.

В дальнейшем изложении материала будем руководствоваться рис. 2.6.

2.5. МЕТОДОЛОГИЯ ФУНКЦИОНИРОВАНИЯ БАЗ ДАННЫХ

Речь пойдет прежде всего о функционировании операционных БД в рамках СУБД.

Независимо от класса БД здесь приходится решать проблемы, к которым относятся обеспечение одно- и многопользовательского функционирования, защита данных, обеспечение целостности, восстановление данных после сбоя в БД.

В *централизованных однопользовательских* БД функционирование обеспечивается так называемыми транзакциями, в результате выполнения которых данные в БД либо обновляются (фиксация), либо остаются прежними (откат). Для операционных БД (СУБД) характерны так называемые короткие транзакции с длительностью в микро- и миллисекунды. Хранилища данных должны работать при длительных (часы) транзакциях. В случае *многопользовательс-*

кого режима дополнительно возникает необходимость одновременного доступа нескольких пользователей к одним и тем же данным, что чаще всего достигается блокировкой данных.

Защита данных от несанкционированного доступа осуществляется либо запретом доступа (пароль), либо разрешением на доступ, что особенно легко обеспечить с помощью языка программирования SQL. Обеспечение целостности определяется специальными программами, получившими название *триггеры*. Они реализуют различного рода ограничения. Например, для поля Пол программа ограничивает задание только значений «муж» и «жен». Другие значения базой данных не воспринимаются.

Восстановление данных после сбоя БД определяется характером сбоя. При кратковременных сбоях БД восстанавливается сама: используются данные БД в контрольных точках и невыполненные транзакции. При длительных сбоях восстановление БД возможно лишь на основе создаваемой и периодически обновляемой резервной копии базы.

В *распределенных одноранговых* БД в решении перечисленных проблем возникают дополнительные сложности. Могут быть использованы распределенные транзакции. Усложняется и процедура одновременного доступа, для обеспечения которой сформированы дополнительные методы. Их суть — та или иная схема централизации управления базой данных. Особую значимость приобретают вопросы дублирования данных.

Новые трудности возникают при интегрировании в неоднородную распределенную БД ранее построенных, действующих локальных баз данных с разными моделями данных.

Решение названных проблем несколько упрощается при использовании в распределенных БД режима клиент—сервер.

Более подробно сведения о функционировании БД приведены в *гл. 4 и 5*.

2.6. МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ ХРАНИЛИЩ ДАННЫХ

Перейдем к методологической стороне хранилищ данных. Здесь по-прежнему можно выделить процедуру создания (проектирования) и использования ХД.

Процедура использования ХД мало отличается от аналогичной процедуры в БД, поэтому ее подробно рассматривать не будем, а отметим одну особенность. В силу высоких требований к быстродействию ХД следует особое внимание обратить на оптимизацию запросов.

При создании ХД выполняются следующие работы.

1. Формируется состав итоговой информации с предельно допустимым временем отклика и предельным сроком хранения детальной информации.

2. Определяется предполагаемый набор запросов на основе детальных данных. При этом следует найти компромисс между созданием итоговой статистической информации и ее вычислениями на основе детальных данных.

3. Выбирается способ хранения данных «время» в таблицах.

4. Осуществляется выбор СУБД, который должен учесть и потребности системы OLTP. Наиболее подходящей является ООСУБД с использованием многомерной модели данных MOLAP. Определяются размерности модели данных.

В то же время можно использовать и реляционную СУБД с применением разновидности ROLAP. Следует выбрать схему («звезда» или «снежинка»). Тогда полезно построить таблицу фактов и сопровождающие ее справочные таблицы с минимальным изменением ключей в них. При использовании схемы «звезда» следует провести денормализацию.

5. Определяются потребности в дополнительных данных, отсутствующих в OLTP, и удаляются ненужные, лишние столбцы в детальных данных.

Контрольные вопросы

1. Назовите требования, предъявляемые к операционным БД; к хранилищам данных.
2. Что такое независимость, безопасность, целостность, защита данных?
3. Как обеспечиваются целостность и независимость данных?
4. Что такое «модель данных» (МД)? Назовите виды МД.
5. Что такое концепция? методология?
6. Расскажите историю развития технологии баз данных.
7. Назовите варианты СУБД.
8. Дайте схематическое представление классического и современного подходов к построению БД.
9. Опишите этапы проектирования централизованной, транзакционной базы данных, хранилища данных.
10. Что такое «хранилище данных»?
11. Каковы специфические требования к ХД?
12. Опишите методологию проектирования ХД.

Глава 3

Общая теория баз данных

С момента появления понятия «база данных» возникла необходимость в теоретической математической поддержке процессов в ней. Постепенно выявились три относительно самостоятельные части теории баз данных: 1) построение баз данных; 2) использование баз данных; 3) функционирование баз данных. Они выстроились в относительно стройную систему позднее в рамках реляционных БД.

В настоящее время при проектировании структур данных применяют три основных подхода.

1. Сбор информации в рамках одной таблицы и последующая ее декомпозиция.

2. Использование CASE-технологии [2].

3. Структурирование информации в процессе проведения системного анализа на основе совокупности правил и рекомендаций.

Первоначально на роль теории баз данных независимо от используемой модели данных претендовала CASE-технология. Позже выяснилось, что CASE-технология охватывает лишь процедуру построения БД, да и то не полностью.

Широкое распространение реляционных БД привело к необходимости добавления в CASE-технологию процедуры нормализации, являющейся частью теории реляционных баз данных, на основе ER-диаграмм. Процедура нормализации была формализована и реализована на компьютере (в диалоговом варианте) в ряде СУБД (Access, Oracle). Вместе с тем, построение ER-диаграмм — процедура специфическая.

Возникают сложности и с процедурой нормализации [2, 3].

В связи с этим реляционные БД чаще проектируют, применяя понятие «отношение».

В то же время в CASE-технологии заложены возможности автоматизации процедуры проектирования баз данных, что особо важно при создании баз данных большой размерности (20 Гбайт и выше).

3.1. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ДАННЫХ

БД, как элемент системы принятия решений (например, в АСУ) есть отражение предметной области реального мира [9]: ее объекты, отношения между ними и отношения в БД должны соответствовать друг другу. Компьютер (и АСУ в частности) оперирует только

формальными понятиями (моделями), соответствующими объектам и связям внешнего мира. В настоящее время имеется свыше тридцати моделей представления данных, которые до последнего времени не были систематизированы.

Их можно разделить на две группы:

1) формальные (математические, скорее теоретические), предполагающие разработку БД обязательно с участием человека;

2) математические представления, рассчитанные на автоматизацию процесса проектирования БД («компьютерное представление»).

Вторая группа рассмотрена в парагр. 3.2, а первую обсудим здесь.

Сразу отметим разницу двух понятий [9]: «модель данных» — средство моделирования; «модель БД» — результат разработки БД. Модель (представление) БД — множество конкретных ограничений над объектами и операциями с ними.

Модель данных (точнее — модель представления данных) есть множество элементов (объектов, типов данных) и связей (отношений) между ними, ограничений (например, целостности, синхронизации многопользовательского доступа, авторизации) операций над типами данных и отношениями.

Множество допустимых типов данных и их отношений образуют структуру данных. В модели данных, следовательно, выделяется три компоненты: структура данных; ограничения, определяющие допустимое состояние БД; множество операций, применяемых для поиска и обновления данных. Эти компоненты отображаются языковыми и программными средствами описания и манипулирования данными.

Описание часто проводят последовательно: структура, ограничения, операции. Начнем с описания структур данных.

Проще всего структуру (отношение) можно задать таблицей с «плоской» (см. табл. 1.11) или сложной (см. табл. 1.12) структурой. При таком задании хорошо видны элементы (столбцы, поля), однако плохо просматриваются отношения, которые могут быть четырех типов: 1:1, 1:M, M:1, M:N.

Более наглядным (особенно для представления типа 1:1) является представление в виде ориентированного графа (рис. 3.1), восходящее к математике, теории автоматического управления и теории информации. Элементами n (n принадлежит N) графа $\Gamma(N, U)$ являются столбцы (поля), а связи между ними определяются дугами u (u принадлежит U). Такому графу соответствует матрица смежности (табл. 3.1) или двудольный граф. Разновидностью графов являются предложенные Д. Мартиным овал-диаграммы (рис. 3.2).

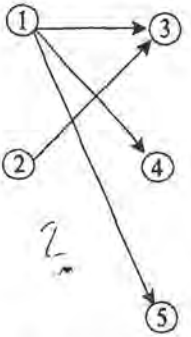


Рис. 3.1. Ориентированный граф:
1—5 — узлы графа

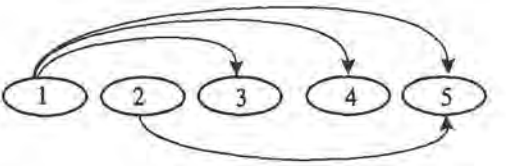


Рис. 3.2. Овал-диаграмма:
1—5 — узлы диаграммы

Таблица 3.1

Матрица смежности

	1	2	3	4	5
1	0	0	1	1	1
2	0	0	0	0	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Теория графов достаточно хорошо развита, однако прямое ее применение для представления данных встречает затруднения, вызванные следующими обстоятельствами:

- связи в моделях представления данных относительно просты (см. рис. 3.1), матрицы смежности получаются разреженными, что снижает ценность их использования;
- в графах отражается чаще всего один тип связи (например, 1:1): выходом здесь может быть использование овал-диаграмм;
- при постановке задачи представления (моделирования) данных, в отличие от теории управления и математики, в которых широко используются начальные предположения, велик объем неформальной составляющей.

Для преодоления третьего затруднения сформировались модели представления данных «сущность — связь» (Entity — Relationship), называемые также «ER-моделями (диаграммами)» или «моделями Чена». Базовыми структурами в ER-модели являются «типы сущностей» и «типы связей».

Отличие типа связи от типа сущности — в установлении зависимости существования реализации одного типа от существования реализации другого. (Например, ЛИЧНОСТЬ — тип сущности, тип СОСТОИТ В БРАКЕ — нет, поскольку реализация последнего типа не существует, если не существует двух личностей. Тип связи может рассматриваться поэтому как агрегат двух или более типов сущностей.)

Выделяют три типа связи: связь «один к одному» (1:1), связь «один ко многим» (1:M), связь «многие ко многим» (M:N).

Примеры этих связей могут быть следующие:

1:1	M:1	M:N (M:M)
студент <—> адрес,	студент <<—> группа,	студент <<—>> преподаватель.

Следует отметить особенности отображения ER-модели. Выделяют следующие типы связей:

- рекурсивное (по «кольцу») множество связей, в котором участвуют несколько сущностей;
- два множества связей между одними и теми же двумя множествами сущностей;
- множество *n*-арных связей, например тернарных (четыре связи, «исходящие от одной сущности»).

Выделение этих связей является крайне важным, так как связи 1:M и M:N имеют внутреннюю неопределенность, что сказывается при операциях модификации. Для преодоления неопределенности на этапе реализации логической модели требуется вводить избыточную информацию.

Отметим, что сущность примерно соответствует таблице, а атрибут — полю реляционной базы данных.

Фрагмент концептуальной модели предметной области «Учебный процесс» представлен на рис. 3.3, а пример представления атрибутов для конкретного объекта показан на рис. 3.4. Выделяют многозначный атрибут, атрибут множества связей.

В общем случае атрибуты отображаются либо на самой ER-диаграмме (при небольшом количестве объектов), либо в виде отдельных приложений по каждому объекту.

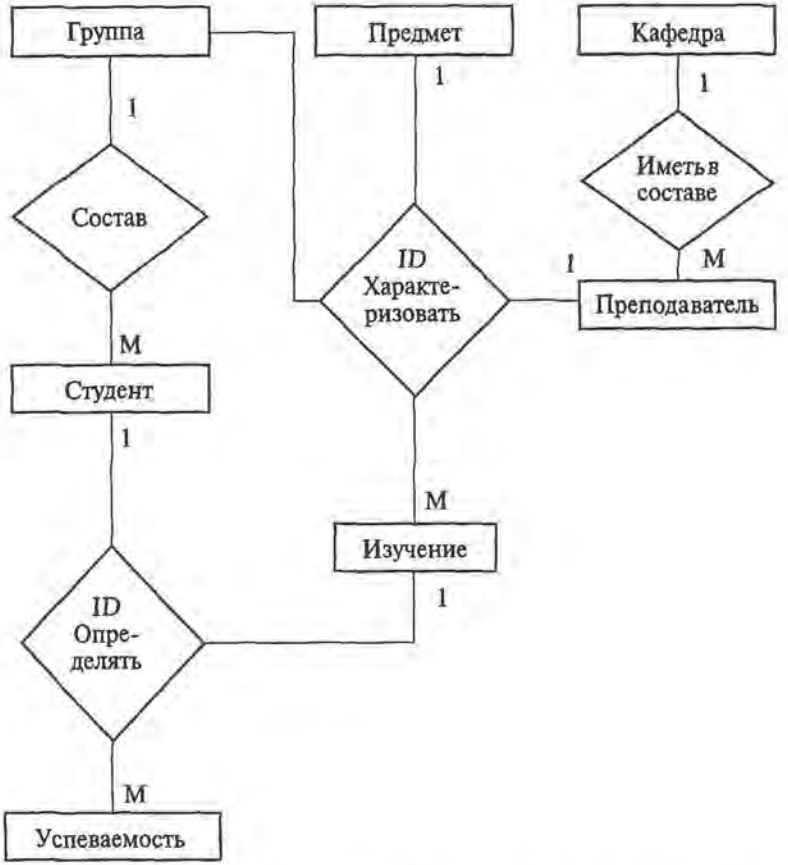


Рис. 3.3. ER-диаграмма предметной области «Учебный процесс»

При построении ER-моделей в ряде случаев целесообразно выделять ряд ограничений:

- ограничение целостности применительно к атрибутам: (например: N — студенты, целое, положительное, число студентов — диапазон от 5 до 35);
- ограничение E по существованию сущностей (рис. 3.3);
- *ID*-зависимость (рис. 3.3): *сущность не может быть идентифицирована в ряде случаев по значениям собственных атрибутов.*

Здесь прямоугольниками показаны типы сущностей и атрибуты, ромбами — типы связей.

Покажем свойства этих моделей на примере БД «Учебный процесс» в институте (рис. 3.4). Укрупненно (и в несколько другом

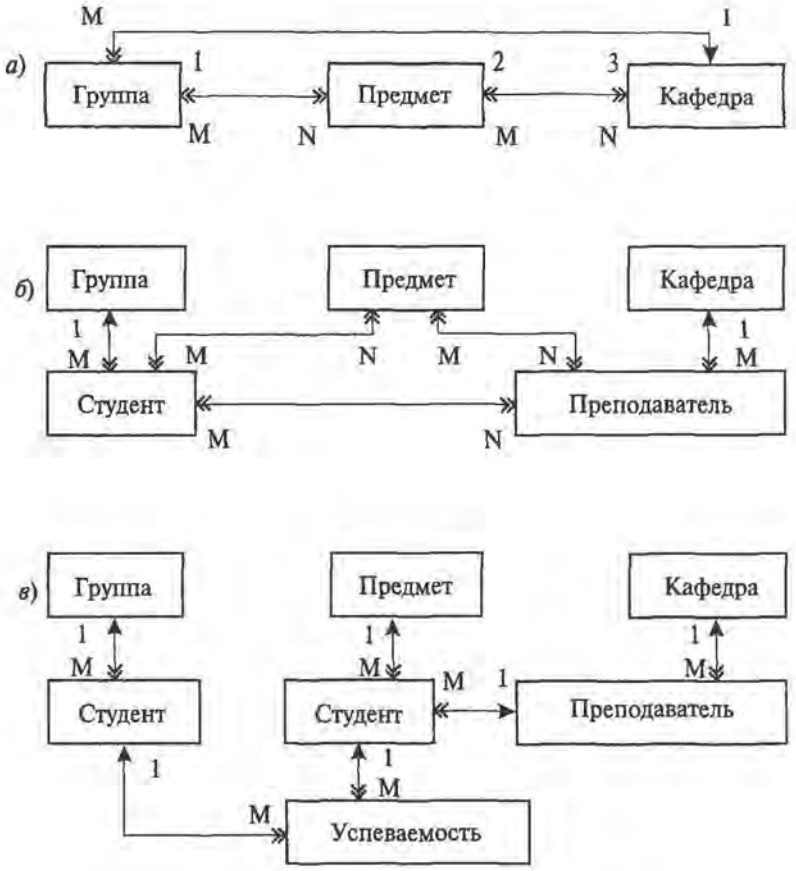


Рис. 3.4. Модель БД «Учебный процесс»:
а — исходная структура; *б* — промежуточная структура; *в* — результат

начертании, чем на рис. 3.3) он может быть представлен в виде отношений трех групп атрибутов (рис. 3.4, *а*) со связями M:N и 1:M. Поскольку группы 1 и 3 — множества, схему можно представить в виде рис. 3.4, *б*. Известно, что ни одна модель данных не может реализовать отношения M:N. В связи с этим схема связей после преобразования окончательно выглядит, как показано на рис. 3.4, *в*.

Заметим, что перечисленные методы обладают следующими недостатками:

• слабо ориентированы на использование компьютеров в проектировании БД;

• оперируют со статическими (неизменными) данными, тогда как в реальных системах управления используются динамические данные (потоки данных);

• отражают потоки данных не системно.

Названные недостатки устранены в CASE-технологии [25].

3.2. CASE-ТЕХНОЛОГИЯ

Для решения задачи автоматизации управления в АСУ возникла необходимость в системном описании процесса управления, включая и принятие решений. Одним из первых моделей в этом направлении были так называемые форрестеровские модели. Позднее появилась методология автоматизации (Structures Analysis Design Technique) SADT, на основе которой построена CASE-технология .

Модельными компонентами CASE-технологии (рис. 3.5) являются составляющие ERD, DFD, STD. Их место в системном описании процесса управления показано на рис. 3.6.

CASE-технология представляет собой систему методов описания, рассчитанную на использование компьютеров при создании БД. Computer-Aided Software/System Engineering (CASE-технология) —

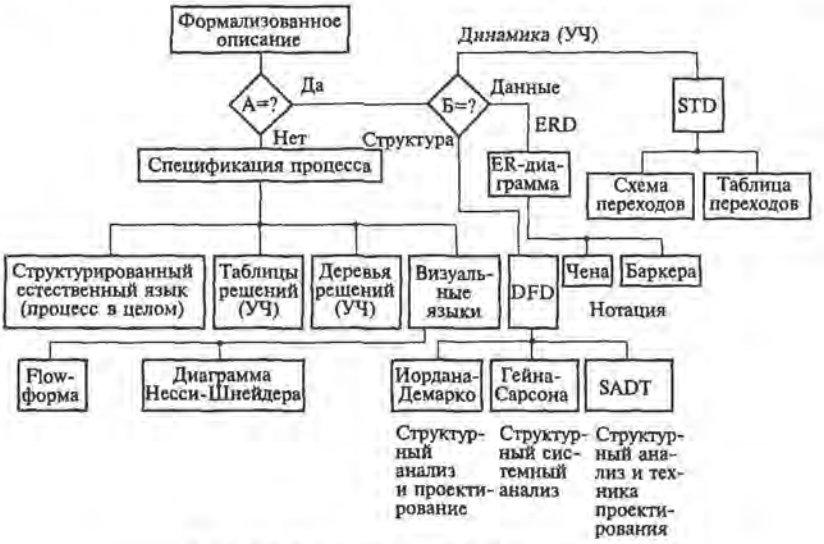


Рис. 3.5. Классификация CASE-методов:

А — элементов много; Б — описание элементов; УЧ — учебный процесс; DFD — Data Flow Diagram; ERD — Entity Relationship Diagram; STD — State Transaction Diagram

Рис. 3.6. Описание процесса в системе

совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем программного обеспечения, поддержанная комплексом взаимосвязанных средств автоматизации. CASE — инструмент для системных аналитиков, разработчиков и программистов.

CASE-технология базируется на методологии системного анализа. Под системным анализом понимают научную дисциплину, разрабатывающую общие принципы исследования сложных объектов и процессов с учетом их системного характера. Его основная цель — сосредоточить внимание на начальных этапах разработки.

В рамках CASE-технологии системный анализ предназначен для отделения проектирования от программирования. В разработке в соответствии с CASE-технологией выделяется построение архитектуры и ее последующая реализация, поэтому системный анализ называют структурным системным анализом или просто структурным анализом.

Важнейшими (базовыми) принципами являются деление (декомпозиция) и последующее иерархическое упорядочение. Они дополняются следующими принципами.

1. Принцип абстрагирования от несущественных деталей (с их «упрятыванием») с контролем на присутствие лишних элементов.

2. Принцип формализации.

3. Принцип концептуальной общности (структурный анализ — структурное программирование — структурное тестирование). Отсюда методология структурного анализа — метод исследования от общего обзора через детализацию к иерархической структуре со все большим числом уровней.

4. Принцип непротиворечивости — обоснование и согласованность элементов.

5. Принцип логической и физической независимости данных.

6. Принцип непосредственного доступа (без программирования) конечного пользователя.

Эта технология положена в основу реализации программных CASE-средств.

Формальным инструментом описания является система диаграмм (см. рис. 3.5): ER-диаграмм (ERD), диаграмм потоков данных (DFD), диаграмм переходов состояний (STD), спецификация процесса, которые обсудим отдельно.

В описании процессов возможны два случая: сложные и простые процессы. Во втором случае разработку БД удобнее вести с использованием понятия «отношение». Рассмотрим поэтому только сложные процессы, учитывая, что CASE-технология для него и разрабатывалась.

ER-диаграммы. Из рис. 3.6 видно, фактически первой разновидностью методов системы CASE-моделей явились ER-модели Чена, подробно рассмотренные в предыдущем параграфе. Здесь отметим лишь ее разновидность — модель Баркера (рис. 3.7). В ней указывается имя сущности, степень множественности (например, 1:M), обязательность (———) или необязательность (.....) связи.

DF-диаграмма. Диаграммы применяются для отображения процессов вход—выход. Первоначально использовалась методология SADT, о которой говорилось ранее, затем перешли на схемы DFD. Применяются две основные разновидности нотаций: Иордана—Демарко и Гейна—Сарсона. Различия между ними невелики и потому используем нотацию Гейна—Сарсона. В нотации используются символы, снабженные именами.

DFD строится на основе декомпозиции и модель верхнего уровня называют *контекстной диаграммой*. В любом конкретном проекте она одна. Такие модели описывают объект управления, а для отражения управляющей части (УЧ) системы применяют [25] расши-

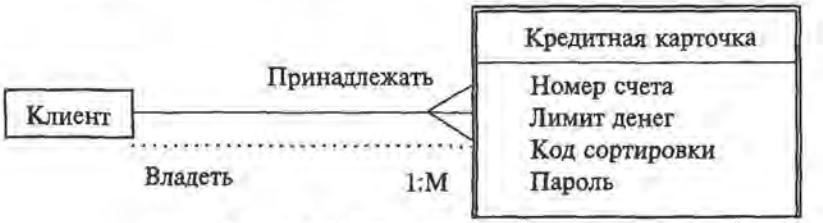


Рис. 3.7. Модель нотации Баркера

Рис. 3.8. Контекстная диаграмма процесса приема на работу

рение реального времени: перечисленные условные обозначения рисуются пунктирными линиями или точками. Основными типами управляющих потоков являются Т-поток (триггер), А-поток (процесс непрерывен, пока поток не выключится), E/D-поток (аналог выключателя с двумя кнопками «включено» и «выключено»).

Иллюстрацию проведем на основе процесса приема на работу, описанного в примере 2.2.

Контекстная диаграмма Гейна—Сарсона представлена на рис. 3.8: она позволяет видеть входные и выходные потоки и внешние сущности (источники и/или приемники данных) «Заказчик» и «Производитель». Детализированная диаграмма рассматриваемого процесса может быть представлена в виде, показанном на рис. 3.9 с процессами

Рис. 3.9. Детализированная диаграмма процесса приема на работу

Рис. 3.10. Расширенная диаграмма процесса приема на работу

1—5, где БД1 — данные или их часть, хранимые в памяти. В общем случае каждый из процессов 1—3, в свою очередь, может быть детализирован. Расширенная диаграмма отображена на рис. 3.10.

Частный случай алгоритма выработки решений, когда число вакансий превышает количество принимаемых, показан на рис. 3.11.

Рис. 3.8—3.11 позволяют заметить, что потоки имеют пояснения. Текстовые средства моделирования получили название словаря данных.

ST-диаграмма. Она используется для отображения процесса выработки и результатов реализации решений. Вводится понятие «состояние». Схематика (схема переходов) для блока «Правила» (рис. 3.11) может быть такой, как показано на рис. 3.12. Процесс изменения состояния может быть отражен с помощью таблицы (табл. 3.2) или матрицы (табл. 3.3).

После рассмотрения деталей CASE-технологии вернемся к системному аспекту. CASE-технологии могут быть классифицированы по нескольким признакам.

1. По шкалам — Software Engineering (SE) и Information Engineering (IE). Первая шкала предназначена для проектирования программного обеспечения и хорошо известна (фактически описана в данной работе), вторая — новая, с более широкой областью применения (для проектирования не только программного обеспечения).

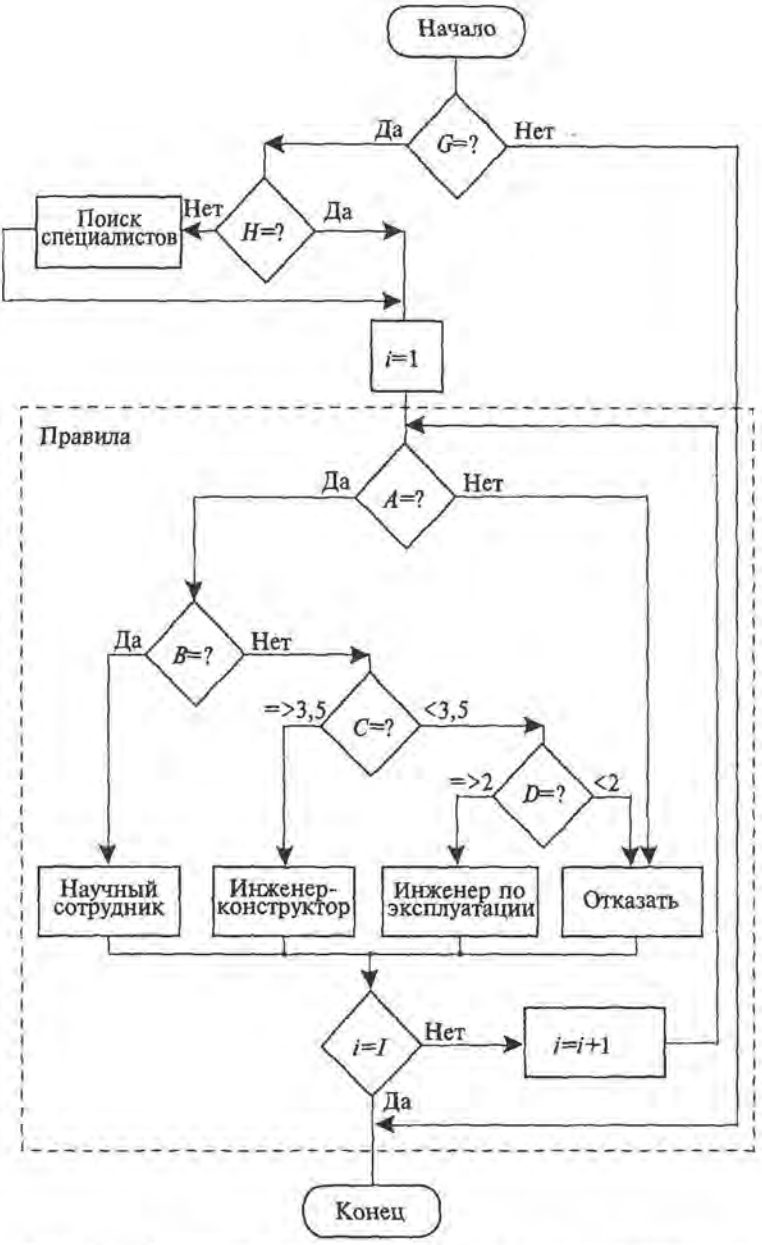


Рис. 3.11. Система правил приема на работу в научное учреждение:

G — вакансия; *H* — претенденты на вакантные должности; *A* — ученая степень; *B* — принимаемый сделал открытие; *C* — средний балл учебы; *D* — опыт работы, лет

Рис. 3.12. STD для процесса принятия на работу;
а — общая схема; *б* — пример

Таблица 3.2

Таблица решений

Текущее состояние	Условие	Действие	Следующее состояние
Начальное состояние	Активизируется в начале каждого сеанса		
Претендент	Правило 1	Отказать	Принимаемый
Претендент	Правило 2	Научный сотрудник	Принимаемый
Претендент	Правило 3	Инженер-конструктор	Принимаемый
Претендент	Правило 4	Инженер по эксплуатации	Принимаемый
Претендент	Правило 5	Отказать	Принимаемый

2. По порядку построения модели: а) процедурно-ориентированный (современный подход); б) ориентированный на данные (традиционный подход).

3. По типу целевых систем — для систем реального времени (управление сложными структурами большого объема данных с интенсивным вводом-выводом) и информационных систем (управление событиями с малым количеством простых по структуре данных с интенсивными вычислениями).

3.3. CASE-СРЕДСТВА

CASE-технология поддерживается, как уже указывалось, CASE-средствами. Интегрированный пакет CASE-средств содержит четыре основных компонента.

Таблица 3.3

Матрица решений

Состояние	Условие	Правило 1	Правило 2	Правило 3	Правило 4	Правило 5
Начальное состояние	Активизируется в начале каждого сеанса					
Претендент		Отказать				
		Принимаемый				
Претендент			Научный сотрудник			
			Принимаемый			
Претендент				Инженер-конструктор		
				Принимаемый		
Претендент					Инженер по эксплуатации	
					Принимаемый	
Претендент						Отказать
						Принимаемый

1. Средства централизованного хранения информации о всем проекте (своеобразная база данных проекта).

2. Средства ввода данных для хранения.

3. Средства анализа, проектирования и разработки.

4. Средства вывода.

Для CASE-технологии характерны четыре основных типа графических диаграмм:

1) функциональное проектирование (DFD);

2) моделирование данных (ERD);

3) моделирование поведения (STD);

4) структурные диаграммы (карты) — отношения между модулями и внутримодульная структура.

CASE-средства (прежде всего фирмы Oracle и отдельных организаций) возможно классифицировать по категориям и по функциональному признаку.

1. По категориям. Выделяют уровень интеграции: вспомогательные программы (tools); пакеты (toolkit); инструментальные средства (workbench, APM).

2. По функциональному признаку. Для анализа и проектирования возможно использовать CASE-аналитик (единственное отечественное средство первого поколения), Application Development Workbench, Easy CASE System Designer.

Проектирование БД существенно упрощается при применении ERWin (фирма Logic Works), Designer/2000 (Oracle), позволяющих проводить логическое моделирование данных, автоматическое преобразование данных в 3НФ.

Программирование (кодогенерирование) — DECACE (DEC), Delphi (Borland).

Сопровождение (поддержка системной документации) и реинжиниринг (анализ, корректировка, реинжиниринг существующей системы) — SuperStructure (Computer Data System).

Управление проектом (планирование, контроль, взаимодействие) — Project Workbench (Applied Business Technology).

Рассмотрим одну из реальных систем автоматизации проектирования БД в рамках Oracle (Cooperative Development Environment — CDE), в которую входят CASE*Dictionary, CASE*Designer, CASE*Generator.

CASE*Dictionary — хранилище информации (БД проекта). CASE*Designer — средство моделирования процессов и данных в системе через внешний интерфейс с помощью средств графического моделирования. CASE*Designer полностью интегрирован с CASE*Dictionary. CASE*Generator на основе информации CASE*De-

signer автоматически генерирует модули программного кода (меню, формы, отчеты). CASE*Generator может генерировать и DLL-сценарии (таблицы, представления, индексы, последовательности) в схеме приложения.

Oracle7 был спроектирован с открытой архитектурой и потому другие компании смогли создать дополняющие средства:

Application Development Workbench (разработка систем на многих платформах) — компания KnowledgeWare;

Easy CASE System Designer (графическое инструментальное средство проектирования, позволяющее генерировать схемы приложения для одной или нескольких СУБД, включая Oracle) — компания Evergreen CASE Tools;

ERWin/ERX (средство проектирования БД для MS Windows) — компания Logic Works;

ADW — интегрированный набор средств для анализа, планирования и моделирования процессов, данных и автоматической генерации приложений.

Несколько иначе выглядит теория реляционных баз данных.

Контрольные вопросы

1. Какие модели представления данных и знаний вы знаете?
2. Что такое CASE-технология?
3. Что такое ERD-, DFD-, STD-составляющие CASE-технологии? Укажите их место в описании системы.
4. Какие Вам известны методы ERD? DFD? STD?
5. Дайте классификацию CASE-технологий, CASE-средств.

Глава 4

Теория реляционных БД

Использование компьютера для реализации какого-либо процесса возможно лишь при наличии теоретического математического описания этого процесса. Сказанное относится и к процессам в базах данных.

В теории реляционных баз данных выделяют представление процедур: а) *создания* БД (с учетом целостности и защиты данных); б) использования базы данных; в) функционирования БД, в том числе при многопользовательском доступе к данным.

При строгом подходе рассмотрение третьей процедуры (как это сделал Э. Кодд) является как бы автономным и не входит в теорию *реляционных БД. Однако, поскольку работа во все более широко* используемом многопользовательском доступе к данным без теоретической проработки процедуры синхронизации невозможна, ее изучение также включим в теорию реляционных БД [4, 12].

Теоретические инструменты первых двух процедур — реляционная алгебра и *реляционное исчисление. Реляционная алгебра (РА) являет*ся теоретической основой алгоритмических языков программирования, *сложных* для начинающего пользователя, тогда как на реляционном исчислении (РИ) построены (см. § 1.1.) более удобные декларативные языки программирования SQL и QBE.

В то же время РА позволяет наглядно отобразить процессы преобразования в базе данных одних таблиц в другие. Именно поэтому рассмотрение *теории начнем с реляционной алгебры.*

4.1. МАТЕМАТИЧЕСКИЕ ОСНОВЫ ТЕОРИИ

Полезно провести аналогию РА и школьного курса алгебры. В последнем в качестве элементов априори вводятся буквы, а операции *над ними — сложение, вычитание (прямые) и умножение*, деление (обратные). В то же время фундаментальная теория алгебры, включающая связи между операциями, такие основные свойства, как коммутативность $ab = ba$, ассоциативность $(ab)c = a(bc)$, *дистрибутивность* $(a + b)c = ab + bc$, *идемпотентность* $a^2 = a$, изучаются в вузовском курсе высшей алгебры.

Сказанное можно отнести и к реляционной алгебре. Ее элементами служат таблицы, над столбцами и строками которых выполня-

ются девять операций (§ 4.1.1.). В то же время более глубокая фундаментальная связь между операциями (свойства), аналогичные курсу высшей алгебры, рассмотрены в § 1.1.

Основы реляционной алгебры

Основные понятия. В основе реляционной БД лежит понятие «отношение», «связь».

Отношением r называется подмножество декартова произведения. Поля отношения (таблицы) могут располагаться в произвольном порядке. Чтобы установить определенный порядок для какой-либо конкретной реализации, вводят понятие «схема» R — множество упорядоченных имен атрибутов $R(A_1, ..., A_n)$. Говорят о схеме R отношения r или r(R). Любому имени A_i ставится в соответствие множество D_i (домен или $dom(A_i)$). Схема — конечное множество кортежей $t \in r$, при этом $t(A_i) = D_i$.

Парадигмой реляционных БД является **ключ r** отношения R — подмножество $K = \{B_1, ..., B_m\}$ R, $m \le n$ с ограничениями:

1) для любых двух кортежей t_1 и t_2 существуют $B \in K$, такое, что $t_1(B) \neq t_2(B)$;

2) $t_1(K) \neq t_2(K)$;

3) ни одно собственное подмножество $K' \subset K$ не обладает свойством ключа.

Ключи, явно перечисленные вместе с реляционной схемой, называются *явными*; в противном случае — *неявными*. Один из выделенных ключей называется *первичным*. Если ключ $K' \subset K \subset R$, то K — суперключ (составной ключ).

Возможен вариант [4], когда несколько минимальных множеств атрибутов функционально определяют все атрибуты отношения. Такие множества называют *возможными* (альтернативными) ключами, поскольку любое из них можно выбрать в качестве составного ключа.

Например, пусть имеется отношение

R(Город, Адрес, Почтовый_индекс).

Очевидно, что атрибуты: Город Адрес → Почтовый_индекс. В то же время Почтовый_индекс → Город (хотя и не адрес). Оба множества могут быть возможными ключами.

Универсум U множество значений всех атрибутов. С учетом ключей схема R над U — совокупность отношений $\{R_1, ..., R_T\}$, где

$$R_i = \{S_i, K_i; 1 \le I \le n\}, \ \sum_{i=1}^{n} S_i = U.$$

Тогда **реляционной БД** d со схемой данных R называется совокупность отношений $\{r_1, ..., r_n\}$, где для любой схемы $R = \{S, K\}$ существует отношение в d, являющееся отношением со схемой S, удовлетворяющее любому ключу.

В реляционной БД, как это видно, оперируют таблицами, простейшим элементом является кортеж (запись).

При проектировании БД (независимо от *применяемого подхода* — традиционного или современного) на входе (источники данных) формируется один состав таблиц и схем отношений, тогда как пользователю может потребоваться совсем другой состав с произвольной схемой (в рамках имеющегося универсума).

Возникает вопрос: каковы должны быть правила создания и преобразования таблиц.

Фактически для таблиц выделяют следующие составляющие:

1) создание;

2) обновление и запрос;

3) синхронизация процессов доступа.

В процессе создания и обновления должна быть обеспечена целостность данных, тогда как в процедуре запроса необходимо преобразование таблиц в *предположении* обеспечения целостности.

Начнем обсуждение с операций преобразования таблиц.

Для оперирования с таблицами необходимо уметь описывать их формально. Теоретическое описание может быть двух видов:

1) предикатами первого порядка;

2) использованием правил реляционной алгебры (РА) и реляционного исчислении (РИ).

Первый вид специфичен и характерен для экспертных (интеллектуальных) систем, поэтому обсудим второй вид.

ОПИСАНИЕ ИСПОЛЬЗОВАНИЯ БД

Алгебра — исходное множество A с определенными на ней операциями вида $f: A^n \rightarrow A$, где n — размерность.

Исчисление — совокупность правил оперирования с какими-либо символами.

РИ по своей сути проще, но его применение ограничено следующими обстоятельствами:

• практически не удается — в рамках РИ — доказать полноту проводимых операций преобразования;

• большая комбинаторная сложность реляционных схем не вскрывается РИ.

Все это возможно с помощью РА, программной реализацией которой, как видно из самой постановки задачи, являются *процедурные* языки программирования.

Реляционная алгебра. Пусть Cl (L) — множество всех замкнутых формул системы L.

Если формула $\varphi \in$ Cl(L), то говорят, что модель M удовлетворяет $\varphi(\varphi \cdot M)$, если φ истинно на M.

Пусть $\gamma \in$ Cl (L). Формула ψ называется следствием γ (*выводима* из γ), если из $\psi \cdot M$ следует $\gamma = M$ для любой модели M.

Любое отношение, построенное правильно с помощью принятой системы операторов и отображений, называется *алгебраическим выражением.* Пусть, по-прежнему, U — универсум (множество атрибутов); D — множество доменов; dom — полная функция из U (dom: $U \to D$); $R = \{R_i, i = 1, p\}$ — множество схем отношений; $d = \{r_i, i = 1, p\}$ — множество всех отношений $r_i(R_i)$; $\theta = \{\neq, \leq, \geq, <, >\}$ — множество бинарных отношений (условий над доменами из D); O — множество операторов (операций), использующих атрибуты из U и отношения из θ.

Реляционная алгебра над U, D, dom, R, d, θ есть семиместный кортеж $B = \{U, D, dom, R, d, \theta, O\}$.

В реляционной алгебре выделяют следующие операции: проекция (обозначается π или P в разных источниках), селекция (σ или S), соединение (J), объединение (U), разность (DF), деление, пересечение, декартово произведение (CP). Пусть имеются два отношения R(A, B, C) и P(D, E, F). Объединения, пересечения и вычитания (разность) выполняются над отношениями одинаковой арности.

1. Операция объединения U(R, P) — без повторений строк:

R(A, B, C) $\cup$ P(D, E, F) = Q(A, B, C)

a	b	c	m	n	o	a	b	c
d	e	f	g	h	i	d	e	f
g	h	i				g	h	i
j	k	l				j	k	l
						m	n	o.

2. Разность (DF(R, P)) — из R удаляются строки, имеющиеся в P:

R(A, B, C) DF P(D, E, F) = Q(A, B, C)

a b c	m n o	a b c
d e f	g h i	d e f
g h l		j k l
j k l		

3. Пересечение R ∩ P — общие элементы множеств:

R(A, B, C) ∩ P(D, E, F) = Q(A, B, C)

a b c	m n o	g h l
d e f	g h i	
g h l		
j k l		

4. Декартово произведение (CP(R, P)): к каждой записи отношения R добавляется каждая запись отношения P:

R(A, B, C) CP P(D, E, F) = Q(A, B, C, D, E, F)

a b c	m n o	a b c m n o
d e f	g h i	d e f m n o
g h l		g h l m n o
j k l		j k l m n o
		a b c g h i
		d e f g h i
		g h l g h i
		j k l g h i

5. Проекция $\pi_{S(A)}(R)$, где S(A) — список доменов результирующего отношения из числа доменов отношения R: выбираются и упорядочиваются столбцы и удаляется избыточность из строк

$R(A, B, C)\ \pi_{B,C}(R) = Q(B, C)$

a	b	c
d	b	c
a	d	k

b	c
d	k

6. Селекция (выбор) $\sigma_F(R)$, где $F(A_i, \theta,$ «константа») — исходное отношение *n*-арности; A_i — атрибут отношения R; θ — логическое условие ($<$, $>$, $=$, $\neq$, $\leq$, $\geq$, $\cap$, $\cup$, $\neg$).

$R(A, B, C)\ Q(A, B, C) = \sigma_{A = a}(R)$

a	b	c
d	b	c
a	d	k

a	b	c
a	d	k.

7. Соединение $J_{AmB}(R, P) = Q = s_{AmB}$:

$R(A, B, C)\ P(D, E)\ Q(A, B, C, D, E)$, где $B \neq D$

a	b	c
d	b	c
a	d	k

d	c
d	a
b	b

a	b	c	d	c
d	b	c	d	c
a	d	k	b	a
a	d	k	b	b.

8. Если сравниваемые поля, имена которых лучше сделать одинаковыми, в результирующем отношении «считаются» только один раз, то говорят о естественном соединении (**слиянии**) NJ:

$$NJ(R, P) = \pi_{1, 2, \ldots, m}(\sigma_{S1, \ldots, Sn})(R \times P),$$

$S_i = (A_i^R = A_i^P;\ I = \overline{1, n})$, A_i — список совпадающих атрибутов в исходном отношении; 1,..., m — упорядоченный список всех компонентов декартова произведения $R \times P$, за исключением $A_1^P, \ldots, A_n^P$.

$R(A, B, C)\ P(D, A, B)\ Q(A, B, C, D)$

a	b	c
d	b	c
a	d	k

d	d	b
a	a	b
c	c	c

a	b	c	a
d	b	c	d
d	b	a	d.

d	b	a

Если отношение состоит из одного кортежа, то при естественном соединении получается селекция.

9. Деление (X, Y) ÷ Y = X.

Операции идут на бинарном (делитель) и унарном (делимое) отношениях, а результат (частное) получается унарным отношением. Элемент x появляется в результирующем отношении, если пара <x, y> имеется в делимом для всех значений элемента y, присутствующих в делителе. Частное — те левосторонние компоненты делимого, чьи правосторонние элементы включают любую компоненту делителя.

Пусть имеется

(X,Y)=A,B)		Y=(B)	X=(A)
1	a	$y_1 = e$	$x_1 = 1$
1	b		3
1	c		4
1	d		5
1	e		
1	f	$y_2 = b$	$x_2 = 1$
2	a	d	4
2	b		
3	c	$y_3 = a$	$x_3 = 1$
3	e	b	
4	b	c	
4	d	d	
4	e	e	
5	e	f	

Наиболее часто используются операции селекции (S), проекции (P) и соединения (J), называемые SPJ-операциями.

Свойства реляционной алгебры

Рассмотрим свойства операций реляционной алгебры [4].

I. Коммутативность:

Унарные операции:

1) $S_{F_1}(S_{F_2}(R)) = S_{F_2}(S_{F_1}(R))$;
2) $P_{A_1}(P_{A_2}(R)) = P_{A_2}(P_{A_1}(R))$, если $A_1 = A_2$;
3) $S_{F_1}(P_{A_1}(R)) = P_{A_1}(S_{F_1}(R))$, если $Attr(F_1)A_1$;
4) $P_{A_1}(S_{F_1}(R)) = S_{F_1}(P_{A_1}(R))$, если $Attr(F_1)A_1$.

Бинарные операции:

5) $J_F(R, S) \to J_F(S, R)$;
6) $U(R, S) \to U(S, R)$;
7) $CP(R, S) \to CP(S, R)$.

II. Ассоциативность бинарных операций:

1) $U(U(R, S), T) \to U(R, U(S, T))$;
2) $CP(CP(R, S), T) \to CP(R, CP(S, T))$;
3) $J_{F_2}(J_{F_1}(R, S), T) \to J_{F_1}(R, J_{F_2}(S, T))$.

III. Идемпотентность унарных операций:

1) $P_{A_1}(R) \to P_{A_2}((P_A(R))$, если $A = A_1$, $A \subseteq A_2$;
2) $S_{F_1}(R) \to S_{F_2}((S_F(R))$, если $F = F_1 \cap F_2$.

IV. Дистрибутивность бинарных операций между бинарными:

1) $S_F(U(R, S)) \to U(S_F(R), S_F(S))$;
2) $S_F(DF(R, S)) \to DF(S_F(R), S_F(S))$;
3) $S_F(CP(R, S)) \to CP(S_F(R), S)$, если $Attr(F) \subseteq Attr(S)$;
4) $S_{F_1}(J_F(R, S)) \to J_{F_1}(S_F(R), S_F(S))$, если $Attr(F) \subseteq Attr(S)$;
5) $P_A(U(R, S)) \to U(P_A(R), P_A(S))$;
6) $P_A(CP(R, S)) \to CP(P_A(R), P_A(S))$.

V. Факторизация унарных операций:

1) $U(S_F(R), S_F(S)) \to S_F(U(R, S))$;
2) $CP(S_{FR}(R), S_{FS}(S)) \to S_F(CP(R, S))$, где $F = FR \cap FS$;
3) $J_F(S_F(R), S_F(S)) \to S_F(J_F(R, S))$, где $F = FR \cap FS$;

4) $DF(S_{FR}(R), S_{FS}(S)) \rightarrow S_{FR}(DF(R, S))$, где $FR \rightarrow FS$;
5) $U(P_A(R), P_A(S)) \rightarrow P_A(U(R, S))$;
6) $CP(P_{AR}(R), P_{AS}(S)) \rightarrow P_A(CP(R, S))$, где $A = AR \cup AS$.

Реляционная алгебра в процедуре использования БД

Перечисленные правила используются прежде всего при формировании и оптимизации запросов.

ОПИСАНИЕ ПОСТРОЕНИЯ БД

Реляционная алгебра больше предназначена для описания процедур запросов (выборки) и обновления. Дело в том, что РА (равно как и РИ, как будет показано позднее) не затрагивают вопросы обеспечения целостности при создании и обновлении БД. Иными словами, они непригодны для того, чтобы СУБД могла автоматически проверить истинность или ложность следствий из заданных в схеме БД [9] ограничений целостности.

Потребовался аппарат с более простой интерпретацией и алгоритмической разрешимостью проблемы выводимости (или логических следствий) разумной комбинаторной сложности.

Это удалось сделать с помощью функционального подхода (формул однозначных {1 : 1, 1 : M} F-зависимостей и многозначных {1 : M и прежде всего — M : M} MV-зависимостей) и, как следствие, путем выполнения процедуры *нормализации*.

F-зависимости обеспечивают переход к первой (1НФ), второй (2НФ), третьей (3НФ) *нормальным формам* представления данных. MV-зависимости позволяют строить четвертую (4НФ) и пятую (5НФ) нормальные формы.

Доказательства положений для F- и MV-зависимостей базируются на реляционной алгебре, в связи с чем их иногда относят к аппарату реляционной алгебры.

Функциональные (однозначные) F-зависимости. Функциональные зависимости являются обобщением понятия ключа: значения кортежа на одном множестве X атрибутов определяют значения на другом множестве Y атрибутов; $X, Y \in R$; R — схема отношения R.

Пусть имеется r(R) и $X, Y \in R$.

Отношение удовлетворяет функциональной зависимости (ФЗ) $X \rightarrow Y$, если $\pi_Y(\sigma_{X = x}(R))$ имеет не более одного кортежа для любого $(\forall)x \in X$. Проверка проводится так (рис. 4.1): если кортежи $t_1(X) = t_2(X)$, то должно удовлетворяться условие $t_1(Y) = t_2(Y)$.

Такие *однозначные зависимости* называются F-зависимостями. Множество F-зависимостей также обозначается F. Иначе сказанное записывается в виде: $F \subset X \to Y$ влечет $X \to Y$ или говорят об аксиоме вывода.

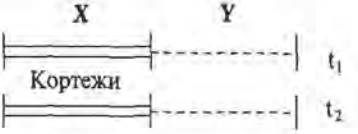


Рис. 4.1. F-зависимость

Аксиома вывода (правило): если отношение удовлетворяет некоторым F-зависимостям, то оно должно удовлетворять и другим F-зависимостям.

Пример 4.1. Пусть имеем расписание занятий «Расписание» (День, Время, Аудитория, Преподаватель, Группа). Введем две ФЗ:

f_1: День Время Аудитория → Преподаватель Группа,

f_2: День Время Преподаватель → Аудитория Группа.

Далее используем следующие ФЗ:

f_1: Студент → Курсовая_работа,

f_2: Студент → Преподаватель,

f_3: Преподаватель → Кафедра,

f_4: Кафедра → Факультет,

f_5: Студент → Факультет.

Пусть $R = \{a, ..., A\}$; $X, Y, Z, W \in R$. Для любого r(R) справедливы следующие аксиомы вывода.

F1. Рефлексивность: $X = X$. Проекция от селекции $\pi_X(\sigma_{X=x}(r))$ имеет не более одного кортежа. Например, Студент → Студент.

F2. Транзитивность: если $X \to Y$ и $Y \to Z$, то $X \to Z$. Если $t_1(X) = t_2(X)$, то $t_1(Y) = t_2(Y)$ по определению. Если $t_1(Y) = t_2(Y)$, то и $t_1(Z) = t_2(Z)$. Следовательно, если $t_1(X) = t_2(X)$, то $t_1(Z) = t_2(Z)$.

Иначе говоря, из Студент → Преподаватель и Преподаватель → Кафедра следует Студент → Кафедра.

F3. Пополнение: *если* $X \to Y$, то $XZ \to Y$.

Из $X \to Y$ следует ($\Rightarrow$), что $\pi_Y(\sigma_{X=x}(R))$ имеет не более одного кортежа для $\forall x \in X$. Если $Z \in R$, то $\sigma_{XZ=xz}(R) \subseteq \sigma_{X=x}(R)$ и $\pi_Y(\sigma_{XZ=xz}(R)) \subseteq \pi_Y(\sigma_{X=x}(R))$ имеет не более одного кортежа.

Следовательно, если Студент → Преподаватель, то Студент Кафедра → Преподаватель.

Или из A → B следует AC → B и AD → B, ABC → B, ABD → B, ACD → B, ABCD → B.

F4. Псевдотранзитивность: если $X \to Y$, $YZ \to W$, то $XZ \to W$.

Если $t_1(X) = t_2(X)$, то $t_1(Y) = t_2(Y)$ по определению. Если $t_1(YZ) = t_2(YZ)$, то и $t_1(W) = t_2(W)$. Следовательно, из $t_1(XZ) = t_2(XZ)$ имеем $t_1(X) = t_2(X)$ и $t_1(Z) = t_2(Z)$, $t_1(Y) = t_2(Y)$, $t_1(YZ) = t_2(YZ)$ и $t_1(W) = t_2(W)$. Иначе, если Студент → Преподаватель, Преподаватель Кафедра → Факультет, то Студент Кафедра → Факультет или из A → B, BC → D следует AC → D.

F5. Аддитивность: если X → Y и X → Z, то X → YZ.

По определению $\pi_Y(\sigma_{X=x}(R))$ и $\pi_Z(\sigma_{X=z}(R))$ имеют не более одного кортежа. Если $\pi_{YZ}(\sigma_{X=x}(R))$ имеет более одного кортежа, то хотя бы одна зависимость имеет более одного кортежа.

Следовательно, Студент → Преподаватель, Студент → Кафедра, то Студент Преподаватель → Кафедра или из A → B, A → C следует A → BC.

F6. Проективность (в определенной степени обратна аддитивности): если X → YZ, то X → Y и X → Z.

По определению $\pi_{YZ}(\sigma_{X=x}(R))$ и $\pi_Z(\sigma_{X=z}(R))$ имеют не более одного кортежа. Однако $\pi_Y(\pi_{YZ}(\sigma_{X=x}(R))) = \pi_Y(\sigma_{X=x}(R))$ и $\pi_Y(\sigma_{X=x}(R))$ тоже имеет более одного кортежа, т. е. X → Y.

Итак, Студент → Преподаватель Кафедра, то Студент → Преподаватель и Студент → Кафедра или если A → BC, то A → C и A → B.

Система F1—F6 является *полной*: с ее помощью можно получить (путем многократного применения) любую F-зависимость для множества F. Аксиомы F2, F5, F6 выводятся из F1, F3, F4. Покажем лишь два вывода.

Транзитивность F2 — частный случай аксиомы F4. Аксиома F5: если X → Y, X → Z, то из F1 следует YZ → YZ, из F4 следует XZ → YZ, из F4 следует X → YZ.

Аксиомы F1, F3, F4 иногда называют *аксиомами Армстронга*.

Замыканием множества F называется множество функций F^+, содержащее все те зависимости, которые могут быть получены применением к F аксиом Армстронга.

Замыкание используется для определения биекции и ликвидации избыточности множества функциональных зависимостей, что можно сформулировать в виде леммы.

Лемма. Функциональная зависимость X → Y выводима из аксиом Армстронга, если $Y \subseteq X^+$.

Доказательство приводится в [4].

Вычисление F^+ достаточно сложная процедура. Например, если $F = \{A_1 \to B_1, \ldots, A_n \to B_n\}$, то F^+ включает A → Y, где Y — подмножества множеств $\{B_1, \ldots, B_n\}$ и их количество составляет 2^n.

Более того, если F-множество F-зависимостей над R и $X \subseteq R$, то в F^+ существует зависимость $X \to Y$ такая, что подмножество Y максимально, т. е. $Z \subset Y$ для любой другой F-зависимости $X \to Z$ в F^+.

Если $F = \{A \to D, AB \to DE, CE \to G, E \to H\}$, то можно показать, что $(AB)^+ = ABCDEH$.

Для $F = \{AB \to C, AE \to G, BC \to D, D \to E, B \to AE, EG \to K\}$ $(AB)^+ = ABCDEGK$.

Сложность алгоритма во времени зависит от размера входного множества F-зависимостей. Следовательно, надо стремиться к минимизации F-множества. Это способствует согласованности и целостности БД (меньше проверок при модификации).

Действительно, множество $F = \{A \to B, B \to C, A \to C, AB \to C, A \to BC\}$ выводимо из множества $G = \{A \to B, B \to C\}$. Говорят, что множество G покрывает множество F (G эквивалентно F или $G \in F$).

На пути минимизации возникает две проблемы: неизбыточность; устранение постороннего атрибута (редуцирование).

Множество F-зависимостей F называется неизбыточным, если нет $F' \subset F$ такого, что $F' \equiv F$. В неизбыточном множестве могут быть посторонние атрибуты, которые могут быть удалены (множество может быть редуцировано).

Пусть имеется множество F-зависимостей $F = \{X_i \to Y_i, i = \overline{1, n}\}$. Множество атрибутов $Z_i \subset Y_i$ называется посторонним в X_i, если $(X_i \backslash Z_i) \to \varphi_i$ может быть получено в замыкании F^+_i.

Множество F-зависимостей F называется **каноническим**, если любая F-зависимость из F имеет вид $X \to A$, а F — редуцировано и неизбыточно.

Множество F-зависимостей F минимально, если оно содержит не более F-зависимостей, чем любое эквивалентное множество F-зависимостей.

Теория F-зависимостей используется при нормализации (1НФ – 3НФ), однако, кроме F-зависимостей, могут существовать MV-зависимости. Для них используются несколько иные правила.

Функциональные многозначные MV-зависимости. Пусть R — реляционная схема; X и Y — непересекающиеся множества на R и $Z = R - (XY)$. Отношение r(R) удовлетворяет многозначной MV-зависимости $X \to\to Y$, если для любых кортежей t_1 и t_2 из r, для которых $t_1(X) = t_2(X)$ в r существует кортеж t_3 такой, что $t_3(X) = t_1(X)$, $t_3(Y) = t_1(Y)$, $t_3(Z) = t_2(Z)$ (*рис. 4.2.*)

Из симметрии существует и кортеж t_4 такой, что $t_4(X) = t_1(X)$, $t_4(Y) = t_2(Y)$ $t_4(Z) = t_1(Z)$.

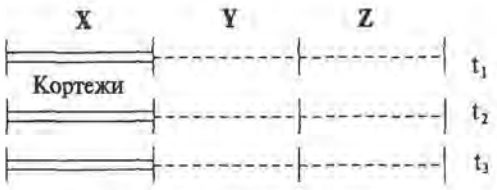


Рис. 4.2. MV-зависимость

Лемма. Если отношение r(R) удовлетворяет MV-зависимости $X \to\to Y$ и $Z = R - (XY)$, то r удовлетворяет $X \to\to Z$, $(X \cap Y) = \varnothing$ (пусто).

Иначе говоря, если существуют кортежи fdp и fd′p′, то должны быть и fd′p, и fdp′.

Теорема. Пусть r — отношение со схемой R; X, Y, Z ⊂ R такие, что $Z = R - (XY)$. Отношение r(R) удовлетворяет MV-зависимости $X \to\to Y$ тогда и только тогда, когда r без потери информации раскладывается в отношения со схемами $R_1 = XY$ и $R_2 = XZ$.

$AB \to\to BC$ и $AB \to\to C$ имеет вид

R	(A	B	C	D)
	a	b	c	d
	a	b	c′	d
	a	b	c	d′
	a	b	c′	d′
	a	b′	c′	d
	a′	b	c	d′

Проверка выполнения $X \to\to Y$ возможна двумя способами.

1. $\{XY\}$ NJ $\{X(R - XY)\} = Z$, где NJ — операция слияния.

2. Пусть $Z = R - (XY)$, $R_1 = XY$, $R_2 = XZ$. Если $r_1 = \pi_{R_1}(r)$, $r_2 = \pi_{R_2}(r)$, то $(r_1)NJ(r_2) \in r$.

Пусть для $x \in X$ существует c_1 кортежей в r_1 и c_2 в r_2 со значением x. Пусть c — число кортежей в r. Если для любого $x \in X = c_1 + c_2$, то $r = (r_1)NJ(r_2)$.

Пример 4.2. Введем обозначения:

$$C_w[X = x](r) = |\pi(\sigma_{X=x}(r))|,\ C_{ABCD}[AB = ab](r) = 4,$$

$$C_C[AB = ab](r) = 2,\ C_D[AB = ab](r) = 2,$$

$$C_{ABCD} = C_C C_D \text{ и } R_1 = ABC;\ Z = R - ABC = D;\ R_2 = ABD.$$

Пусть R — схема отношений и X, Y, Z, W $\subset$ R.

Для MV-зависимостей существует ряд аксиом, которые по своему виду несколько отличны от аксиом F-зависимостей.

M1. Рефлексивность: $X \to\to X$.

M2. Пополнение: если $X \to\to Y$, то $XZ \to\to Y$.

M3. Аддитивность: если $X \to\to Y$ и $X \to\to Z$, то $X \to\to YZ$. Эти аксиомы похожи на аксиомы F-зависимостей.

M4. Проективность: если $X \to\to Y$ и $X \to\to Z$, то $X \to\to Y \cap Z$, $X \to\to Y - Z$.

M5. Транзитивность: если $X \to\to Y$ и $Y \to\to Z$, то $X \to\to Z - Y$.

M6. Псевдотранзитивность: если $X \to Y$ и $YW \to\to Z$, то $XW \to\to Z - YW$.

Следующая аксиома не имеет аналога для F-зависимостей.

M7. Дополнение: если $X \to\to Y$ и $Z = R - (XY)$, то $X \to\to Z$.

Докажем только аксиому M3, поскольку остальные доказательства похожи по своей идее.

Из $X \to\to Y$ следует существование кортежа t_1 со свойствами $t_3(X) = t_1(X)$, $t_3(Y) = t_2(Y)$, $t_3(V) = t_2(V)$, где $V = R - (XY)$.

Аналогично из $X \to\to Z$ следует $t_4(X) = t_1(X)$, $t_4(Z) = t_1(Z)$, $t_4(W) = t_2(W)$, где $W = R - (XZ)$.

Надо найти кортеж $t(X) = t_1(X)$, $t(YZ) = t_1(YZ)$, $t(U) = t_2(U)$, где $U = R - (XYZ)$. Пусть $t = t_4$. Тогда $t(X) = t_4(X)$, $t_4(Z) = t(Z)$, $t_4(Y) = t_4(Y \cap W) = t_3(Y \cap W) = t_1(Y \cap W) = t(Y \cap W)$ и $t_4(YZ) = t(YZ)$. Поскольку $U \subseteq (R - XY) \cap (R - XZ) = V \cap W$, то $t_4(X) = t_4(U) = t_3(U) = t(U)$.

Когда речь идет об MV-зависимостях, то вместе с ними могут существовать и F-зависимости. F- и MV-зависимости связаны двумя аксиомами.

C1. Копирование: если имеется r(R) и $X \to Y$, то $X \to\to Y$.

C2. Объединение: если $X \to\to Y$ и $Z \to\to W$, где W Y и $Y \cap Z = \phi$, то $X \to\to W$.

Можно показать, что:

1) система M1—M7 полна;

2) система F1—F6, M1—M7, C1—C2 для множеств F- и MV-зависимостей полна.

Свойства F- и MV-зависимостей используем в § 4.2 в процедуре нормализации.

В ней участвуют две составляющие:

1) F-зависимости, которые дают возможность получить 1НФ, 2НФ, 3НФ, а также нормальную форму Бойса—Кодда (НФБК);

2) MV-зависимости, связанные с 4НФ и 5НФ.

F-зависимости. Основное правило: необходимо, чтобы r(R) разлагалось на отношения (схемы), например, R_1 и R_2, без потерь, т. е.

$$\{\pi_{R_1}(r)\} \text{NJ} \{\pi_{R_2}(r)\} = r.$$

Нормализация возможна через декомпозицию или методом синтеза.

Рассмотрим основные положения процедуры декомпозиции, наиболее часто используемый в практике.

Говорят, что F-зависимость $X \to Y$ применима к схеме R, если $X \to Y$ является F-зависимостью над R. Пусть БД $d = \{r_1, ..., r_n\}$ со схемой $R = \{R_1, ..., R_n\}$ и F-множество F-зависимостей над U. БД d удовлетворяет F, если любая F зависимость $X \to Y$ из F^+ применима к схеме R и выполняется отношение r.

В общем случае F может быть избыточно и потому выявляют $G \subseteq F$ и $G \equiv F$. Говорят, что G навязан схеме R и используют G-зависимость для композиции.

Пусть G-множество всех F-зависимостей в F^+, которые применимы к какой-либо схеме $R_i \in R$. Любая F-зависимость в G^+ называется **навязанной** R, любая F-зависимость (F^+—G^+) — ненавязанной R. Множество F навязано схеме R БД, если любая G-зависимость в F^+ навязана R, т. е. $G \in F$.

Пример 4.3. Пусть $R = \{R_1, R_2, R_3\}$, $R_1 = ABC$, $R_2 = BCD$, $R_3 = DE$ и $F = \{A \to BC, C \to A, A \to D, D \to E, A \to E\}$. $A \to D$ и $A \to E$ неприменимы ни к одной схеме R_i. Однако F навязано R, как $G = \{A \to BC, C \to A, C \to D, D \to E\} \equiv F$ и каждая F-зависимость в G применима к некоторой схеме в R. $F' = \{A \to D\}$ не является навязанной.

Основы реляционного исчисления

РА реализуется в виде процедурных языков, но те же процессы запроса и обновления могут быть описаны на языке реляционного исчисления (РИ). В реляционном исчислении выделяют исчисление доменов и исчисление кортежей.

Исчисление кортежей при прежних обозначениях определяется шестеркой

$$<U, D, dom, R, d, Q> \qquad (4.1)$$

и имеет выражение {x(R)|f(x)}, где f — разрешенная формула над U; x — свободная переменная.

Исчисление кортежей — формализованная система обозначений и правил для записи выражения нового правила.

Для описания операций в исчислении кортежей возможно использовать предложенный Э. Коддом язык АЛЬФА. В нем [9] используются следующие операторы: get w — получить из рабочей области w; range — область; hold — найти (первичный ключ запоминает место кортежей); put — включить; having — принадлежащий; update — обновить; insert — вставить; delete — уничтожить; up, down — (сортировка) по возрастанию, убыванию; count — считать (число записей); total — общий итог. Применяются следующие обозначения: Y, X — множество элементов данных из домена X в реляционном отношении Y; ∃ — существует; ∀ — любой; ∪ — ИЛИ; ∩ — И; ⌉ — НЕ; «x» — символ значения x.

Запросы в РИ выражаются путем спецификации предиката, которому должны удовлетворять кортежи и домены. Формулы в РИ строятся из атомов и совокупности логических и алгебраических операций.

Каждый пользователь обладает своей рабочей областью w (область связи между пользователем и моделью данных). В рабочую область w попадают данные, отобранные в результате поиска (рис. 4.3). Рабочая область w содержит отношения, выделенные из МД посредством оператора get. Отношение имеет форму таблицы.

После завершения работы оператора get содержимое w может обрабатываться любым способом, который допускают базовые языки программирования. Оператор get обеспечивает всегда прямой поиск (по значениям, условиям для данных) последовательно, но не по адресу. Единицей доступа в языке РИ является экземпляр логической записи (кортеж). Пользователь может определить в операторе get любую выборку доменов из одного или нескольких отношений.

На основе исчисления кортежей построен декларативный язык программирования *Structured Query Language (SQL)*.

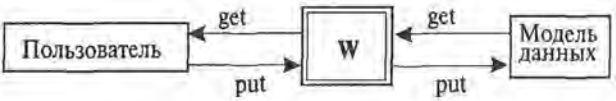


Рис. 4.3. Схема выполнения запроса

Пример 4.4. Иллюстрацию основных операций на языке РИ будем проводить на следующих отношениях [9]:

СТУД(СН, СФ, Балл, Курс, Группа, Кафедра),
ПРЕП(ПН, ПФ, Должность, Кафедра, Зарплата, Надбавка),
СТПР(СН, ПН, Часы),

где СТУД, ПРЕП, СТПР — отношения Студент, Преподаватель, Студент—Преподаватель; СН, СФ — номер зачетной книжки и фамилия студента соответственно; ПН, ПФ — табельный номер и фамилия преподавателя соответственно.

Вариантов описания запросов очень много, поэтому рассмотрим их основные классы.

I. Операторы поиска:

а) простой поиск — получить фамилии студентов (избыточные дублирующие значения не помещаются в рабочую область w)

get w(СТУД.СФ);

На языке SQL ему соответствует команда

SELECT СТУД.СФ FROM Студ;

Более подробно язык SQL рассмотрен в гл. 5.

б) поиск с использованием кванторов ∃ и ∀ и логических операций — существуют ли фамилии студентов такие; табельный номер преподавателя — 2486

range СТПРx
get w(СТУД.СФ): ∃x (X.СН = СТУД.СН∩X.ПН = '2486');

II. Операции обновления:

а) простое обновление — заменить название кафедры ИиУС у преподавателя с табельным номером 2486

hold w (ПРЕП.ПН, ПРЕП.Кафедра): ПРЕП.ПН = 2486 w.Кафедра = ИиУС
update w;

б) простое включение — включить (put) в отношение запись о студенте с номером зачетной книжки 857 Петрове: балл — 3, курс — 4, группа И4, кафедра ИиУС

w.СН = 857
w.СФ = Петров
w.Бал 3
w.Кур с= 4
w.Группа = И4
w.Кафедра = ИиУС
put w(СТУД);

в) удаление — удалить данные о студентах, имеющих балл менее 3

hold w(СТУД): СТУД.Балл < 3
delete w;

д) обновление основного ключа — заменить табельный номер преподавателя с № 785 на № 127

hold w(ПРЕП): ПРЕП.ПН = 785
delete w
w.ПН = 127
put w(ПРЕП).

III. Вычисляемые операции (библиотечные функции):

а) количество записей о студентах

get w(count(СТУД.СН));

Исчисление доменов. В исчислении доменов, определяемом той же шестеркой элементов (4.1), переменные используются на доменах и операции могут содержать несколько переменных.

Запрос: найти номера, фамилии и баллы студентов 4 курса группы И4 кафедры ИиУС

{X Y Z | СТУД (X Y Z 4 И4 ИиУС)}.

Запрос: получить фамилии преподавателей и количество часов

{X Y P | ∃Z ∃L ∃M ∃N ∃Q(ПРЕП(Z ИиУС M N X P) ∩ СТПР(Q Z Y))}.

В общем виде — это выражения

$$\{a_1 \ldots a_n \mid (\exists b_1) \ldots (\exists b_m)\ (R_1(c_{11} \ldots c_{1k})) \ldots (R_p(c_{p1} \ldots c_{pk}))\},$$

где c_{ij} — некоторые значения a_i, b_i, появляющиеся не менее одного раза, или константы. Все переменные неявно связаны квантором

существования и отображаются подчеркнутыми символьными строками. Неподчеркнутые строки являются константами. Вывод на печать помечается символом P.(print), за которым возможно указание имени переменной (например P.A.).

На основе исчисления доменов М.М. Злуфф, как отмечено в [9], создал [9] визуальный декларативный язык программирования Query By Example (QBE).

В силу своей универсальности язык SQL (точнее SQL2) получил широкое распространение в современных реляционных БД и будет подробнее рассмотрен в гл. 5.

В свете сказанного закономерно возникает вопрос о сопоставлении РА и РИ в смысле полноты и замкнутости операций преобразования таблиц.

Общим для РА и РИ (рис. 4.4) является простота, независимость данных от физической организации и физических методов доступа.

Достоинством реляционной алгебры является возможность доказательства полноты системы операций. Однако операции реляционной алгебры соответствуют процедурным языкам, что характерно для разработчиков и пользователей-программистов и не очень удобно для пользователей-непрограммистов. Операция реляционного исчисления реализуется в виде декларативных языков программирования, что для пользователя предпочтительнее. Иными словами, РА дополняется РИ. В то же время доказательство полноты системы операций в реляционном исчислении затруднительно.

Доказано [4, 10, 11] однозначное соответствие групп операций реляционной алгебры и реляционного исчисления. Это одновременно является доказательством полноты и замкнутости системы операций в РИ и возможности перехода к декларативным языкам типа SQL.

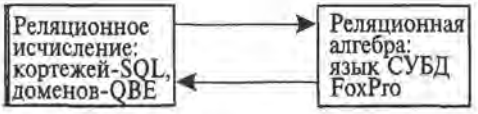


Рис. 4.4. Соотношения реляционного исчисления и реляционной алгебры

4.2. ПОСТРОЕНИЕ БД

Построение БД предполагает построение структуры, ее заполнение данными, определение области доступа к данным и их защита.

Построение структуры связано с нормализацией.

Нормализация. В процессе нормализации — на основе рассмотренного формального аппарата F- и MV-зависимостей строятся 1НФ—5НФ.

1НФ является плоским файлом: в каждом поле может быть только атомарная запись, запись списка или другого сложного объекта не допускается.

Первая нормальная форма **1НФ**. Отношение находится в первой нормальной форме, если значения всех его атрибутов простые (*атомарные*), т. е. значение атрибута не должно быть множеством или повторяющейся группой. В школьном курсе вводится алгебра: элементы — поля N, операции — сложение, вычитание, умножение, деление. Их основные свойства: коммутативность ab = ba, ассоциативность (ab)c = a(bc), дистрибутивность (a + b)c = ab + bc, идемпотентность $a^2 = a$.

Ненормализованному отношению соответствует многоуровневая таблица (иерархия) в отличие от однородной табличной структуры нормализованного отношения.

Проведем построение 1НФ—3НФ на основе примера 4.4.

Пример 4.5.

СТПР	(СН,	ПН,	Кафедра,	Факультет,	Часы),
	s1	p1, p6	k1	f1	10, 10
	s2	p1	k1	f1	20.

Очевидно, что таблица СТПР не находится в 1НФ, поскольку в полях ПН и Часы имеют место списки. Приведем таблицу к 1НФ.

СТПР	(СН,	ПН,	Кафедра,	Факультет,	Часы),
	s1	p1	k1	f1	10
	s1	p6	k1	f1	10
	s2	h1	k1	f1	20.

Рассмотрим операции обновления (включение, удаление, обновление).

Включение. Невозможно поставить номер зачетки студента СН, если не известен табельный номер преподавателя ПН.

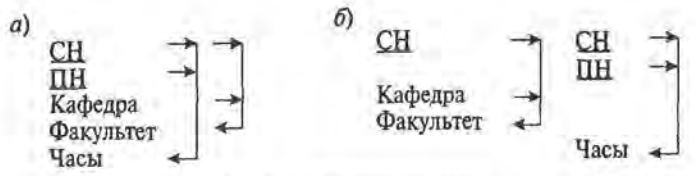


Рис. 4.5. Построение 2НФ

Удаление. Удаление часов удаляет остальную информацию.

Обновление. Изменение k1 требует просмотра нескольких записей.

Для устранения этих недостатков строится 2НФ (рис. 4.5).

Вторая нормальная форма обеспечивает полную зависимость неключевых атрибутов от ключевых. Полная зависимость означает, что в любой момент времени каждому значению атрибута A соответствует не более одного значения атрибута B, связанного с ним отношением R.

Из рис. 4.5, *а* видно, что полной зависимости от составного ключа СН ПН нет и таблица распадается на две (рис. 4.5, *б*).

С (СН,	Кафедра,	Факультет)	СЧ (СН,	ПН,	Часы)
s1	k1	f1	s1	p1	10
s2	k1	f1	s2	p1	20.

Здесь можно ввести информацию о студенте, если отсутствует номер преподавателя; удалить часы, не удаляя данных о студенте; обновить название кафедры только один раз. Из двух таблиц легко восстанавливается одна таблица. Заметим, что количество хранимой информации уменьшилось.

Однако и здесь, в таблице С, имеют место недостатки.

Включение. Если нет СН, нельзя ввести данные о факультете и кафедре.

Удаление. Удаление данных k1 влечет удаление данных о студенте.

Обновление. Изменение зависимостей «Кафедра—Факультет» потребует просмотра многих записей.

Необходима 3НФ. Третья нормальная форма устраняет избыточные зависимости между неключевыми атрибутами. Зависимость «Кафедра—Факультет» является транзитивной и удаляется, а таблица (рис. 4.6, *а*) распадается на две: ФС и КС (рис. 4.6, *б*).

ФС (СН,	Факультет)	КС (Факультет	Кафедра)
s1	f1	f1	k1.
s2	f1		

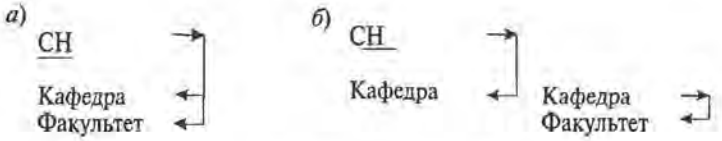


Рис. 4.6. Построение 3НФ:
а — общая таблица; *б* — таблицы ФС и КС

Существует нормальная форма Бойса—Кодда (НФБК). Схема отношения R находится в НФБК относительно F-зависимости F, если для любого $Y \subseteq R$ и любого атрибута $A \subset R \backslash Y$ из $Y \to A$ следует $Y \to R$, т. е. если Y нетривиально определяет произвольный атрибут R, то ключ Y есть суперключ K.

Если Y — не ключ в R, то $R = (R \backslash A, YA)$.

НФБК существует не всегда.

3НФ является окончательной, если имеют место зависимости 1 : 1 и 1 : M. Для отношений M : M процедура продолжается путем построения 4НФ и 5НФ для MV-зависимостей.

Пусть F-множество F- и MV-зависимостей над U. Схема отношения R находится в 4НФ относительно F, если для каждой MV-зависимости $X \to\to Y$, выводимой из F и $XY \subset R$ либо MV-зависимость тривиальна, либо X — суперключ для R. Схема БД R находится в 4НФ, если каждая входящая в нее схема отношения находится в 4НФ относительно F.

MV-зависимость $X \to\to Y$ называется тривиальной для схемы R, содержащей XY, если любое отношение r(R) удовлетворяет $X \to\to Y$.

Пример 4.6. Отношение «Сбыт» (табл. 4.1) удовлетворяет требованиям MV-зависимостей и может быть представлено в виде 4НФ: каждая из подсхем (табл. 4.2, 4.3) находится в 4НФ.

Таблица 4.1

Отношение «Сбыт»

Завод	Товар	Магазин
z_1	t_1	m_1
z_1	t_1	m_2
z_1	t_2	m_1
z_1	t_2	m_2
z_2	t_2	m_2

Таблица 4.2

Отношение «Производство»

Завод	Товар
z_1	t_1
z_1	t_2
z_2	t_2

Таблица 4.3

Отношение «Снабжение»

Завод	Магазин
z_1	m_1
z_1	m_2
z_2	m_2

Здесь возможна и другая трактовка (рис. 4.7): отношение M : M заменяется двумя отношениями 1 : M с помощью промежуточного ключа, которым оказывается поле «Завод». Нетрудно видеть, что схема табл. 4.1 обладает свойством соединения таблиц (табл. 4.2 и табл. 4.3) без потерь. Фактически отношение «Сбыт» находится в 5НФ.

В то же время отношение «Сбыт1» (табл. 4.4) не отвечает требованиям MV-зависимостей и не может быть разложено на схемы «Производство» и «Снабжение», поскольку, как легко видеть, при соединении из них не восстанавливается схема «Сбыт1».

Отношение находится в 5НФ, если его полная декомпозиция содержит во всех проекциях возможный ключ (ключ-кандидат). Иными словами, отношение R разлагается на несколько схем, каждая из которых находится в 4НФ.

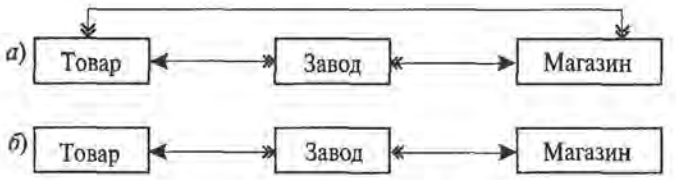


Рис. 4.7. Промежуточный ключ

Таблица 4.4

Отношение «Сбыт1»

Завод	Товар	Магазин
z_1	t_1	m_1
z_1	t_1	m_2
z_1	t_2	m_1
z_2	t_2	m_2

4.3. ИСПОЛЬЗОВАНИЕ БД

Запросы, как показано ранее, могут быть описаны на языках реляционной алгебры и реляционного исчисления.

С помощью реляционной алгебры может быть построено дерево запроса. Более того, запрос может быть оптимизирован [4, 9, 12, 13]. Оптимизация может выполняться с учетом следующих правил.

1. Выполнение селекции как можно раньше.
2. Сортировка файлов или создание индексов путем соединения декартова произведения с последующей селекцией.
3. Предварительное вычисление общих выражений.
4. Выполнение селекции и проекции за один просмотр.
5. Комбинирование проекции с предшествующими или последующими операциями.
6. Объединение селекции с предшествующим декартовым произведением.

Алгоритмически это может быть представлено в таком виде.

Шаг 1. Селекцию $S_{F_1} \cap ... \cap S_{F_N}(E)$ представить как каскад селекций $S_{F_1}(...(S_{F_N}(E)))$ — закон I.4 (см. § 4.1).

Шаг 2. Переместить любую селекцию в дереве как можно ниже — законы I.4, IV.1, IV.3, IV.5.

Шаг 3. Переместить любую проекцию в низ дерева — законы I.2, IV.5, IV.6.

Шаг 4. Скомбинировать любой каскад селекций (проекций) в одиночную селекцию (проекцию) — законы I.1, I.2, I.5, — или селекцию с последующей проекцией.

Шаг 5. Разбить внутренние узлы дерева на группы: объединить двухместные операции с предшествующими или последующими узлу унарными операциями S и P.

Шаг 6. Перейти к более высокому уровню иерархии.

Пример оптимизации. Пусть дана [4] база данных «Библиотека»:

КН[ИГИ](Название[_книги], Автор, Название-издательства, N_бк),
ИЗД[АТЕЛИ](Название_издательства, Место, Адрес_издательства),
ЧИТ[АТЕЛИ](Фамилия, Адрес, Город, N_форм[уляра]),
ВЫД[АЧИ](N_форм[уляра], N_бк, Дата),

где бк — библиотечный каталог, а для обозначения таблиц и полей используем сокращения (без букв в квадратных скобках).

Запрос: найти, у кого находятся книги, выданные до 01.01.1997.

Ему соответствует на языке АЛЬФА запрос (рис. 4.8, *а*)

$$\pi_{\text{НАЗВАНИЕ}}\,\sigma_{\text{ДАТА} \le \text{1.1.1997}}\,\pi_S(\sigma_F(\text{ВЫД, ЧИТ,КН})),$$

где F = ЧИТ.N_форм = ВЫД.N_форм ∩ КН.N_бк = ВЫД.N_бк, S = Название_книги, Автор, Название_издательства, N_бк, Фамилия, Адрес, Город, N_форм, Дата).

На рис. 4.8, *а* разделяют две селекции и перемещают как можно ниже в дереве. Селекция

$$\sigma_{\text{ДАТА} \le \text{1.1.1997}} \tag{4.2}$$

ниже проекции и двух селекций по законам I.1, I.5.

Селекция (4.2) применяется к произведению (ВЫД × ЧИТ) × КН. Дата — единственный атрибут отношения ВЫД, потому

$$\sigma_{\text{ДАТА} \le \text{1.1.1997}}((\text{ВЫД} \times \text{ЧИТ}) \times \text{КН})$$

на

$$(\sigma_{\text{ДАТА} \le \text{1.1.1997}}(\text{ВЫД} \times \text{ЧИТ})) \times \text{КН}$$

и

$$((\sigma_{\text{ДАТА} \le \text{1.1.1997}}(\text{ВЫД})) \times \text{ЧИТ}) \times \text{КН}.$$

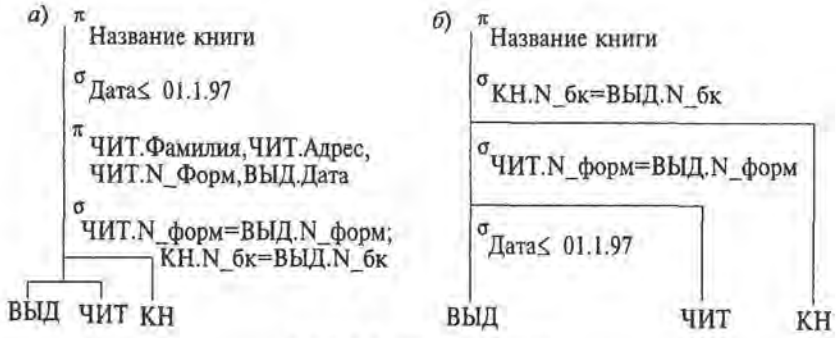
Селекцию можно переместить вниз по дереву. Селекция с условием КН.N_бк = ВЫД.N_бк не может быть перемещена ниже любого декартова произведения, поскольку имеет атрибуты как отношения КН, так и других отношений.

$$\sigma_{\text{ЧИТ.N\_ФОРМ}} = \sigma_{\text{ВЫД.N\_ФОРМ}}$$

может быть перемещена ниже и применена к произведению

$$\sigma_{\text{ДАТА} \le \text{1.1.1997}}((\text{ВЫД}) \times \text{ЧИТ}.$$

ВЫД.N_форм есть имя атрибута отношения, полученного в $\sigma_{\text{ДАТА} \le \text{1.1.1997}}$(ВЫД), ибо это атрибут отношения ВЫД.



Р и с. 4.8. Дерево запроса

По закону I.2 две проекции комбинируются в одну $\pi_{\text{НАЗВАНИЕ}}$ и результат отражается на рис. 4.8, *б*.

По закону I.4 $\pi_{\text{НАЗВАНИЕ}}$ и $\sigma_{\text{кн.N\_бк = выд.N\_бк}}$ заменим на каскад

$$\pi_{\text{НАЗВАНИЕ}},\ \sigma_{\text{кн.N\_бк = выд.N\_бк}},\ \pi_{\text{НАЗВАНИЕ, кн.N\_бк, выд.N\_бк}} \qquad (4.3)$$

По закону IV.6 выражение (4.3) заменяется на $\pi_{\text{НАЗВАНИЕ, кн.N\_бк}}$, примененного к отношению КН и

$$\pi_{\text{выд.N\_бк}}, \qquad (4.4)$$

примененного к левому оператору декартова произведения более высокого уровня (рис. 4.8, *б*).

Последняя проекция (4.4) взаимодействует с нижней селекцией по закону I.4 и получается каскад:

$$\pi_{\text{выд.N\_бк}},\ \sigma_{\text{чит.N\_форм = выд.N\_форм}},\ \pi_{\text{выд.N\_бк, чит.N\_форм, выд.N\_форм}}. \qquad (4.5)$$

Проекция (4.5) «просеивается» через декартово произведение по закону IV.6 и частично — через $\sigma_{\text{ДАТА} \le 1.1.1997}$(ВЫД) по закону I.4. Кроме того, проекция $\pi_{\text{выд.N\_бк, выд.N\_форм, дата}}$ — излишняя (это все атрибуты ВЫД) и исключается. Тогда окончательный результат принимает вид, приведенный на рис. 4.9, на котором группы операторов обведены пунктирными линиями. Любое из декартовых произведений является фактически эквисоединением, если его скомбинировать с селекцией, находящейся в дереве выше. Иными словами, селекция для ВЫД и проекция для ЧИТ могут быть скомбинированы в группы I и II, при этом сначала выполняются операции группы I, а затем — группы II.

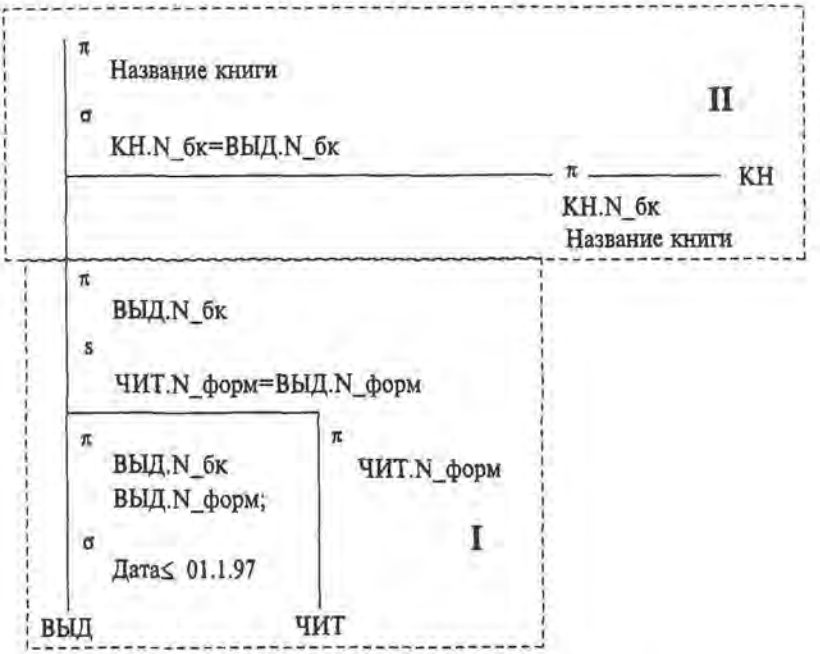


Рис. 4.9. Преобразованное дерево запроса

Полученный результат может оказаться неоптимальным, поскольку критерий оптимизации как таковой отсутствует.

Процедура оптимизации выполнения запроса реализована в ряде СУБД. Так, в СУБД Access [29] она выполняется через последовательное обращение к элементам меню Сервис\Анализ\Быстродействие. Далее необходимо ввести индексы для всех полей составного первичного ключа, указать все поля с критериями выбора или сортировки, связей между таблицами.

4.4. ФУНКЦИОНИРОВАНИЕ БД

Рассматривая процесс функционирования БД, обеспечиваемый, как правило, СУБД, следует сказать о синхронизации работы отдельных блоков, безопасности (целостности и защите) данных, восстановлении данных БД после сбоя.

Синхронизация имеет место как при однопользовательском, так и при многопользовательском режимах. Первоначально поговорим об **однопользовательском режиме**, обеспечиваемом транзакциями.

Под транзакцией понимается:

• входное сообщение, передаваемое в систему и отражающее некоторое реальное событие в компьютере (БД);

• процесс изменения в БД, вызванный передачей одного входного сообщения.

К транзакции предъявляются следующие основные требования:

• она выполняется полностью или не выполняется совсем;

• транзакция должна иметь возможность возврата, при этом независимый возврат в начальное состояние до момента изменения состояния всех объектов;

• транзакция должна быть воспроизводима: при воспроизводстве блокировку необходимо осуществлять до момента просмотра всех объектов.

Возможна двухфазная и трехфазная схема транзакции. Последняя более сложная и применяется в ограниченном объеме (например, в системе управления распределенной БД SDD-1). Поговорим о широко применяемой двухфазной схеме транзакции.

Первая фаза (операция фиксации): все устройства, участвующие в транзакции, сообщают о своей готовности специальной программе — системному журналу (журналу транзакций).

Вторая фаза по системному журналу транзакций происходит выполнение или возврат в прежнее положение (откат) в точки фиксации (точки контроля) (рис. 4.10).

Возврат (восстановление) может быть оперативным, промежуточным, длительным. Для двух первых устанавливается предыдущее состояние (точки фиксации), для последнего — контрольные точки. Контрольные точки предназначены для процедуры восстановления БД после серьезного сбоя и устанавливаются периодически (например, с интервалом 2—3 секунды). Интервал определяется при настройке БД или компьютера.

Перейдем к **многопользовательскому режиму** (рис. 4.11). Здесь задача синхронизации процессов усложняется за счет взаимодействия нескольких пользователей.

Для любой транзакции ТР возможны две группы действий:

• изменение состояния (запись) — W;

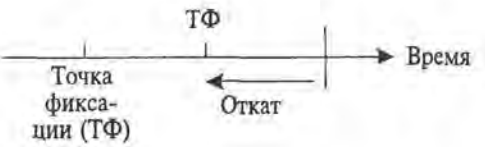


Рис. 4.10. Действие транзакции

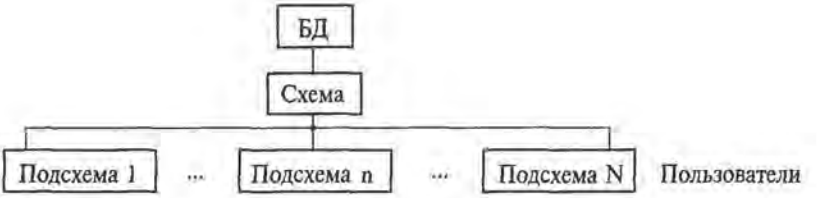


Рис. 4.11. Многопользовательский режим

• считывание (чтение) состояния — R.

Для двух взаимодействующих транзакций TP_1 и TP_2 возможны следующие случаи.

1. Одновременное изменение TP_1 и TP_2, при этом возможны наложение данных (потеря обновления) или ошибка в первой транзакции (взаимозависимость восстановления).

2. Изменение R_1 и считывание W_2 с двумя возможными последствиями:

а) изменение с помощью TP_1 значения, считываемого TP_2 (воспроизводимость считывания);

б) откат перед окончанием работы TP_1 (поскольку нет изменений в TP_2) и возврат TP_1 в прежнее состояние.

3. Чтение TP_1 и изменение TP_2 (нет гарантии воспроизводимости).

Для устранения этих нежелательных явлений возможны такие способы управления:

• блокировка монопольная или согласованная;

• отложенные изменения;

• привязка по меткам времени работы компьютера (временная привязка).

Второй и третий способы рассмотрим при изучении распределенных баз данных, а здесь остановимся на первом способе.

Блокировка выполняется только для одной транзакции и одного объекта.

При последовательном (во времени) выполнении транзакций нарушений целостности нет, но такое выполнение требует много времени. В связи с этим используют так называемый параллельный режим (параллелизм). Вводится понятие «правильно оформленная транзакция» [4].

1. Перед обработкой объекта должна быть выполнена его блокировка.

2. После обработки объект должен быть разблокирован.

3. Перед разблокировкой не должна выполняться повторная блокировка.

4. Неблокированный объект не должен освобождаться.

Для параллельного выполнения транзакций составляется расписание (определяющее порядок их взаимодействия). Поскольку последовательное расписание обеспечивает целостность, «параллельное» расписание делают эквивалентным последовательному (последовательностно-подобные или сериальные расписания).

Очевидно, что в монопольной блокировке возможны ожидания и рестарты (перезапуски ожидающих транзакций). При этом возможны тупики (рис. 4.12, *а*): одна транзакция блокирует другую и процедура выполнения транзакций «зависает». Для выявления таких тупиков транзакции отражают графически (рис. 4.12, *б*).

Для выхода из тупиков при монопольной блокировке возможны следующие варианты:

а) любая транзакция единовременно запрашивает все нужные блокировки на начальной стадии своего выполнения;

б) откат и рестарт, если время ожидания больше установленного порогового значения;

в) рестарты с приоритетом по времени.

При использовании согласованных блокировок возможно:

- предупреждение о блокировке для всей области;
- блокировка части области, связанной с данной транзакцией.

Возможны и другие виды блокировок (голосование по большинству, метод предварительного анализа конфликтов), рассматриваемые далее применительно к распределенным базам данных.

Безопасность. Напомним, что безопасность — защита БД от умышленного и неумышленного искажения информации и нарушения секретности. Защита от неумышленного искажения осуществляется с помощью ограничения целостности, от преднамеренного искажения — управлением доступом.

Ограничения целостности. Выделяют две группы ограничений.

Рис. 4.12. Монопольная блокировка (тупик)

I. В процессе проектирования: 1) при получении достоверных данных из источников; 2) при построении структуры; 3) при заполнении БД данными (в том числе ссылочная целостность).

II. При эксплуатации: 1) машинные сбои; 2) ошибки оператора.

Поскольку ограничения группы II характеризуют больше СУБД, чем БД, они рассмотрены в гл. 5.

При построении структуры и заполнении ее данными имеет место целостность отдельных таблиц и ссылочная целостность (триггеры). Реализация триггерных условий на основе реляционной алгебры осуществляется программным путем, например, в Access.

Использование реляционного исчисления (например, в СУБД InterBase) позволяет применять объектно-ориентированный подход (интерфейс пользователя и алгоритм приложения), декларативный язык и задавать с помощью СУБД при формировании структуры триггеры (в качестве условий), уникальность ключей, связи таблиц (на их схеме с помощью мыши). Может быть непосредственно использован язык SQL.

Защита. Управление доступом противодействует несанкционированному доступу в цепочке «субъект — данные — процедуры».

Простейшим способом является введение системы паролей, т. е. идентификации пользователей. Обычно они используются в сетевом режиме при входе в сеть. Этот способ прост, но для БД недостаточно удобен, поскольку пароли все же можно и узнать.

В дальнейшем использовали таблицы и список управления доступом. Этот метод тоже недостаточно гибок.

В последнее время, особенно после появления языка SQL, стали широко использовать гибкую систему разрешений и запрещений доступа к данным. Для ограничения доступа возможно использовать и виды, описанные в гл. 5.

Контрольные вопросы

1. Что такое реляционная алгебра? реляционное исчисление?
2. Опишите математическое соответствие реляционной алгебры и реляционного исчисления.
3. Какие операции реляционной алгебры вы знаете? Какие из них наиболее часто используются? Какими типами языка они реализуются программно?
4. Какие разновидности реляционного исчисления вам известны? На какую программную реализацию они «выходят»?
5. Что такое «запрос по примеру (QBE)»?
6. Какие группы операций языка SQL вы знаете?
7. Можно ли считать язык SQL универсальным языком реляционных СУБД?

Глава 5

Реляционные базы данных

Освоение свойств моделей данных (МД) осложняется тем, что в разных моделях данных используется своя терминология описания структурных элементов и их связей. Для упрощения изучения МД ниже приведены соответствующие термины в сравнении с понятиями, используемыми в реляционной модели данных.

Краткая характеристика структуры моделей данных

Модель данных	Элемент структуры	Связь	Структура таблицы
Реляционная	Таблицы: столбцы — поля,строки — записи	По ключу	Линейная
Иерархическая	Сегменты: исходный и порожденный — аналоги таблиц	По указателю	Линейная
Сетевая	Записи: владелец и член — аналоги таблиц	По указателю и по ключу (связь именуется); совокупность записей и связь образуют набор	Линейная, нелинейная
Объектно-реляционная	Объекты (таблицы,абстрактные типы данных)	По ключу	Линейная, нелинейная
Объектно-ориентированная	Классы объектов (типов данных, данных): объект — строка,столбцы — свойства (константы,встроенные объекты,потоки данных,коллекции,многомерные переменные,ссылки)	По объектной ссылке и объектному указателю	Линейная, нелинейная

При обсуждении моделей данных (гл. 5—9) будем придерживаться такой однотипной последовательности:

1) логическая структура, включающая описание структуры (элементы, связи);

2) создание БД в рамках рассматриваемой модели данных (МД);

3) использование БД;

4) свойства модели данных (достоинства и недостатки). Сравнительная таблица свойств МД приведена в гл. 9.

5.1. ЛОГИЧЕСКАЯ СТРУКТУРА

Реляционные базы данных получили широкое распространение в персональных компьютерах. Наиболее известны такие локальные СУБД, как dBASE, Paradox и особенно — Access. СУБД Oracle, Sybase, Informix, BTrieve, Ingress, InterBase были изначально предназначены для работы в сети с большими объемами данных.

В основе реляционной модели лежит математическое понятие теоретико-множественного отношения, которое представляет собой подмножество декартова произведения списка доменов.

Домен — множество значений (например, множество целых чисел). Декартовым произведением доменов D_1, D_2, .., D_k (обозначается как $D_1 \times D_2 \times ... \times D_k$) называется множество всех кортежей $(V_1, V_2, ..., V_k)$ длины k, таких, что $V_i \in D_i$, $i = \overline{1, I}$. Например, если $k = 2$, $D_1 = \{0, 1\}$ и $D_2 = \{a, b, c\}$, то $D_1 \times D_2$ есть {(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)}, а отношением может быть, например, {(0, a), (0, c), (1, b)}.

Элементы отношения называются кортежами и имеют арность k (степень отношения), причем i-й компонентой является V_i. Отношение удобно представлять таблицей — совокупностью всех кортежей: каждая строка есть кортеж и каждый столбец соответствует одному компоненту. Кортежи обычно нумеруются и их количество определяет размерность таблицы. Столбцы называются атрибутами, и им часто присваиваются имена. Упорядоченный список имен атрибутов отношения называется схемой отношения. Если отношение называется «Студент» и его схема имеет атрибуты A_1, A_2, ..., A_k, то такую схему будем записывать как СТУДЕНТ(A_1, A_2, ..., A_k).

Совокупность схем отношений называется схемой (реляционной) БД, а текущие значения соответствующих отношений — БД.

Данные из диаграммы объектов-связей представляются двумя видами отношений.

1. Набор объектов может быть представлен отношением, содержащим все атрибуты данного набора объектов. Если объекты набора идентифицируются с помощью связи с другим объектом, то схема отношения содержит дополнительно атрибуты ключа второго набора.

2. Связь между наборами объектов E_1, E_2, ..., E_k представляется отношением, схема которого состоит из атрибутов ключей каждого из этих наборов.

Реляционная модель есть представление БД в виде совокупности упорядоченных нормализованных отношений.

Для реляционных отношений характерны следующие особенности.

1. Любой тип записи содержит только простые (по структуре) элементы данных.

2. Порядок кортежей в таблице несущественен.

3. Упорядочение значащих атрибутов в кортеже должно соответствовать упорядочению атрибутов в реляционном отношении.

4. Любое отношение должно содержать один или более атрибутов, которые вместе составляют уникальный первичный ключ.

5. Если между двумя реляционными отношениями существует зависимость, то одно отношение является исходным, второе — подчиненным.

6. Чтобы между двумя реляционными отношениями существовала зависимость, атрибуты, служащие первичным ключом в исходном отношении, должны также присутствовать в подчиненном отношении.

Э. Кодд первоначально предложил 12 (дюжину) правил, фактически требований, которым должна удовлетворять реляционная база данных.

1. Правило информации. Вся информация на логическом уровне представляется только значениями в таблицах без использования указателей и индексов.

2. Правило гарантированного доступа. Каждый атомарный элемент таблицы доступен через комбинацию из имени таблицы, имени поля и ключа.

3. Системная поддержка Null-значений. Для представления отсутствующих данных с любым типом используют Null-значение.

4. Динамический оперативный каталог на основе реляционной модели. Метаданные (словари) формируются теми же языками, что и данные.

5. Правило исчерпывающего подъязыка данных. В базе данных возможно использовать несколько языков программирования, однако один (чаще всего — SQL) должен быть главным.

6. Правило обновления представления (вида, View). Все теоретически обновляемые представления может обновлять и система.

7. Ввод, обновление и удаление данных на высоком уровне. Работа с несколькими записями должна быть характерна не только запросам на выборку, но и запросам на обновление.

8. Физическая независимость данных. Возможно изменение адреса БД, изменение физической компоновки БД не оказывая влияния на работу прикладных программ и пользователя.

9. Логическая независимость данных. При добавлении или удалении элементов (таблиц, полей) в структуре БД другие части базы данных остаются неизменными.

10. Независимость целостности. Ключ не должен иметь значения Null. Первичный и родительский ключи должны быть уникальными. Каждому значению внешнего ключа должно существовать значение родительского ключа. Ссылочная целостность уменьшает быстродействие из-за проверки условий-связей через словарь.

11. Независимость распределения. В распределенной БД расположение данных независимо. Для пользователя такая БД должна выступать как централизованная БД.

12. Правило соблюдения правил. *Нельзя обходить ограничения*, введенные с помощью языка SQL.

Пример 5.1. Представим БД «Учебный процесс» в виде реляционной модели (табл. 5.1). Далее отношения (например, табл. 5.1, *а*) будем записывать и в другой форме:

ГРУППА(Шифр группы, Название, Количество {студентов}, Средний_балл).

Подчеркнутый атрибут является ключевым.

Таблица 5.1

а) Отношение «Группа»

Шифр_группы	Название	Количество	Средний балл
1	И1	16	4,3
2	И2	23	4,0
3	И3	18	4,2

б) Отношение «Студент»

Номер_зач_кн	Шифр_группы	ФИО студента	Год рождения	Средний балл
И-1746	1	Серов А.П.	1979	4,1
И-1747	2	Киров П.Г.	1980	4,0
И-1748	3	Сухов П.Н.	1981	4,5

в) Отношение «Кафедра»

Код кафедры	Название	Телефон	Зав. кафедрой
1	ИиУС	154-12-86	Сорокин П.В.
2	АПП	171-12-05	Борисов Б.В.
3	ТПП	212-10-81	Степанов И.В.

г) Отношение «Преподаватель»

Табел_номер	ФИО преподавателя	Уч_степень	Уч_звание	Код кафедры
381	Шаталов А.С	д. т. н.	Профессор	1
101	Сидоров А.Т.	к. т. н	Доцент	2
402	Тараканов П.Т.	к. ф.-м. н	Доцент	3

д) Отношение «Предмет»

Код предмета	Название	Всего часов	Практ/лаборатор	Семестров
П1	Информатика	350	130	2
П2	Кибернетика	300	120	3
П3	Математика	600	200	4

е) Отношение «Изучение»

Шифр_группы	Код предмета	Табел_номер	Вид занятий	Часы
2	П2	402	Практические	
2	П2	381	Лекции	
1	П3	381	Лекции	
1	П3	401	Практические	

ж) Отношение «Успеваемость»

Шифр_ группы	Номер_ зач_кн	Код предмета	Табел_ номер	Вид занятий	Оценка
1	И-1746	П3	381	Экзамен	5
2	И-1747	П3	381	Экзамен	4
3	И-1748	П3	381	Экзамен	3

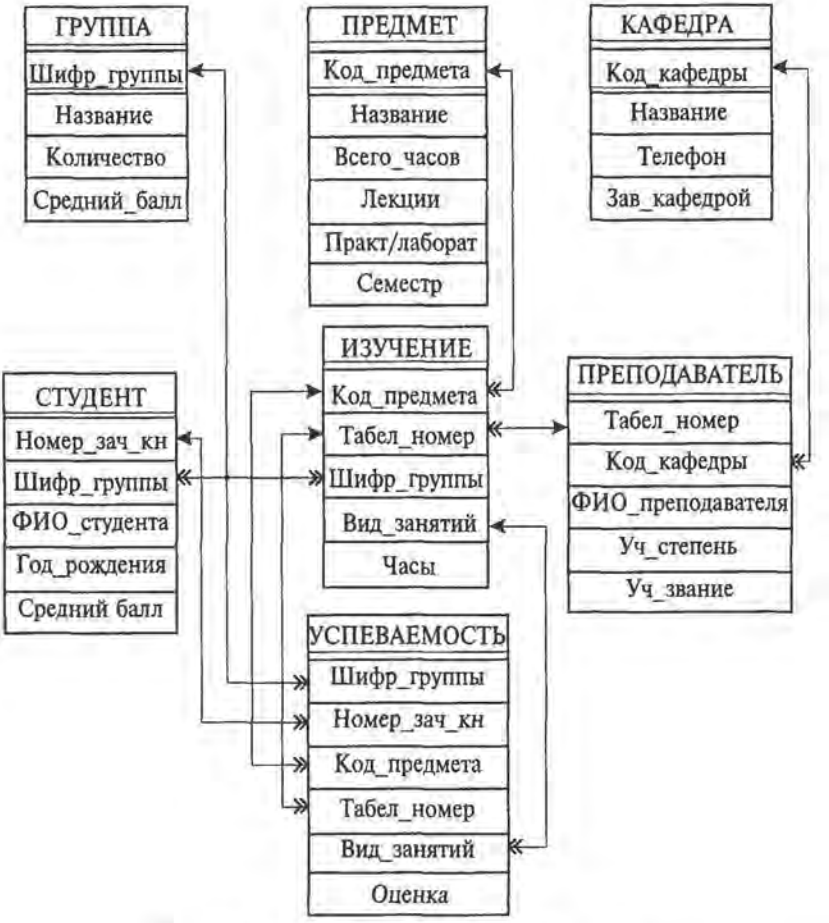


Рис. 5.1. Схема связей БД «Учебный процесс»

Процедуры создания и использования реляционных БД основываются на теории реляционных БД, подробно рассмотренной в гл. 4. Ее результаты используем в прикладных целях. При введении структуры данных используют соответствующие форматы данных. Для таблицы «Преподаватель» они представлены в табл. 5.2. Вся БД (табл. 5.1) представлена в 4НФ, поэтому отразим схему связей между ее отношениями (рис. 5.1), где подчеркнутые поля — первичные ключи. Уточненный (в процессе проектирования) перечень таблиц и полей с их форматами данных приведен в приложении 1, а схема связей — в гл. 5.

Таблица 5.2

Форматы типов данных отношения «Преподаватель»

Имя поля	Ключ	Уникальное поле	Обязательное поле	Тип данных	Размер	Подписи поля
Табел_номер	Первичный	Да	Да	Числовой	Целое	Таб N
ФИО преподавателя	-	Нет	Да	Текстовый	30	ПФИО
Уч_степень	-	Нет	Нет	Текстовый	12	Уч_ст
Уч_звание	-	Нет	Нет	Текстовый	12	Уч_зв
Код кафедры	Внешний	Нет	Да	Текстовый	6	Код_каф

5.2. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ БД

Основная задача при проектировании реляционных БД — формирование оптимальных отношений.

Для формализации процесса построения оптимальной реляционной БД используется теория нормализации, основанная на том, что определенный набор отношений обладает лучшими свойствами при включении, модификации и удалении данных, чем все остальные наборы отношений, с помощью которых могут быть представлены те же данные (см. гл. 4).

Нормализация осуществляется последовательно с использованием пяти нормальных форм, включая форму Бойса—Кодда.

На этом «бумажное» построение БД заканчивается. Компьютерная реализация БД определяется языками описания (ЯОД) и манипулирования (ЯМД) данными. Они могут базироваться на реляционной алгебре (процедурные языки) и реляционном исчислении кортежей и доменов (декларативные языки). На исчислении кортежей основан язык SQL, на исчислении доменов — язык QBE.

Рассмотрим возможности языков SQL и QBE. Прикладное их использование описано в гл. 15.

Язык SQL

Для SQL имеется много вариантов и диалектов. Здесь изложим основные положения базового варианта: более подробное описание языка приведено в [2, 5, 6, 26, 33].

Иллюстрацию языка SQL проведем на примере базы данных «Снабжение», представленной в табл. 5.3—5.5. Для нее схема Access-связей показана на рис. 5.2.

Таблица 5.3

«Продавцы»

пном	пимя	город	комм
1001	Строков	Москва	12
1002	Кирюшин	Пермь	13
1004	Аврорин	Москва	11
1007	Удалов	Курск	15
1003	Козлов	Орел	10

Таблица 5.4

«Заказчики»

зном	зимя	город	рейтинг	пном
2001	Иванов	Москва	100	1001
2002	Петров	С.-Петербург	300	1003
2003	Ковров	Сочи	200	1002
2004	Силоров	Кириши	300	1003
2006	Крабов	Русса	100	1001
2008	Конкин	Пермь	300	1007
2007	Красин	Луга	100	1004

Таблица 5.5

«Заказы»

прном	сумпр	датпр	зном	пном
3001	18.69	10.03.1990	2008	1007
3003	767.19	10.03.1990	2001	1001
3002	1900.10	10.03.1990	2007	1004
3005	5160.46	10.03.1990	2003	1003
3006	1098.16	10.03.1990	2008	1007
3009	1713.23	10.04.1990	2002	1003
3007	75.75	10.04.1990	2004	1002
3008	4783.00	10.05.1990	2006	1001
3010	1309.95	10.06.1990	2004	1002
3011	9891.88	10.06.1990	2006	1001

Название полей таблиц

Таблица «Продавцы»:

пном — уникальный номер продавца, первичный ключ;

пимя — имя продавца;

город — город расположения продавца;

комм — комиссионные продавца.

Таблица «Заказчики»:

зном — уникальный номер заказчика, первичный ключ;

зимя — имя заказчика;

рейтинг — число, показывающее уровень предпочтения данного заказчика перед другими;

пном — номер продавца, внешний ключ.

Таблица «Заказы»:

прном — уникальный номер заказа, первичный ключ;

сумпр — сумма (цена) заказа;

датпр — дата получения заказа;

зном — номер заказчика, делающего заказ, внешний ключ;

пном — номер продавца, продающего заказ, внешний ключ.

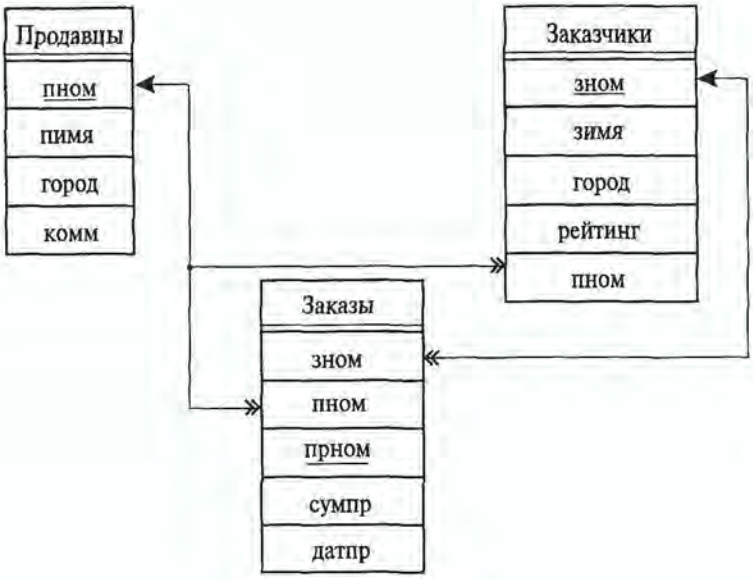


Рис. 5.2. Схема связей

Выделяют статический и динамический языки программирования SQL. В *статическом языке* значения всех объектов четко зафиксированы. В *динамическом языке* SQL вместо значений объектов используются параметры, данные для которых или вводятся в диалоговом режиме пользователем, или заимствуются в процессе выполнения программы из других таблиц (баз данных).

Статический язык программирования SQL

В языке SQL возможно выделить три основные группы операций: создание (CREATE), обновление (INSERT, UPDATE, DELETE), запрос (SELECT). Они имеют следующие стандарты, в которых приняты обозначения: | — все, что предшествует символу, можно заменить тем, что следует за ним; {} — единое целое для применения символа; [] — необязательное выражение; ... — повторяется произвольное число раз; .,... — повторяется произвольное число раз, но любое вхождение отделяется запятой.

```
CREATE TABLE <имя таблицы>
                ({<имя столбца> <тип данных> [размер]
                [<тип столбца> ...]}.,...);
                [<тип таблицы>].,..);                    (5.1)
```

Типы данных могут быть: INTEGER, CHARACTER, DECIMAL, NUMERIC, SMALLINT, FLOAT, REAL, PRECISION, LONG, VARCHAR, DATE, TIME. Четыре последние типа не входят в стандарт SQL, но им могут поддерживаться.

Тип столбца (и тип таблиц) может быть: UNIQUE, PRIMARY KEY, CHECK <предикат>, DEFAULT = <список полей>, REFERENCE <имя таблицы> [(<имя столбца>.,...)].

```
INSERT INTO <имя таблицы>[(<имя столбца>.,...)]
    {VALUES(<список полей>).,...}                    (5.2)

DELETE <имя предиката>
    [WHERE <имя предиката>
    |WHERE CURRENT OF <имя курсора>
    (*только вложенный*)];                           (5.3)

SELECT * |[{DISTINCT|ALL]<список полей>
    FROM <имя таблицы> [<алиас>]} ...
    [WHERE <предикат>]
    [GROUP BY {<имя столбца>|<целое>}.,...]}
    [HAVING <предикат>] [ORDER BY {<имя столбца>}]|<целое>}...]
    [{UNION}];                                       (5.4)
```

Выделяют три разновидности языка SQL: интерактивный, вложенный и встроенный.

Интерактивный SQL используется для функционирования непосредственно в базе данных, чтобы производить вывод для использования его заказчиком.

Встроенный SQL состоит из команд SQL, помещенных внутри программ, которые обычно написаны на другом языке (типа КОБОЛА или Паскаля). Это делает такие программы более мощными и эффективным.

Будем рассматривать преимущественно (при отсутствии упоминаний) интерактивный язык.

♦ ИНТЕРАКТИВНЫЙ ЯЗЫК SQL

В нем возможно выделить:

• DDL (Язык Описания Данных) — язык описания схемы и в ANSI он состоит из команд, создающих объекты (таблицы, индексы, виды) в базе данных;

• DML (Язык Манипулирования Данными) — набор команд, определяющих, какие значения представлены в таблицах в любой момент времени;

• DCD (Язык Управления Данными) состоит из средств, которые определяют разрешение пользователю выполнять определенные действия.

По своей сути язык SQL является специфическим декларативным языком запроса, в связи с чем наибольшее число комбинаций существует для процедуры SELECT. Однако более удобно расположить команды по технологическому циклу работы с БД:

1) создание БД — структуры таблиц, создание видов, заполнение БД данными, обеспечение целостности, система доступа (разрешений), словарь данных, многопользовательский режим;

2) использование БД — запрос в различных формах (в том числе с обновлением).

Создание БД

Структура таблиц. Таблицы (пустые) создаются командой CREATE TABLE (выражение (5.1)).

```
CREATE TABLE Продавцы
    (пном integer,
     пимя char (10),
     город char (10),
     комм decimal);                                    (5.5)
```

Команда CREATE TABLE в основном определяет имя таблицы, набор имен столбцов, указанных в определенном порядке, типы данных и размеры столбцов.

Таблицы принадлежат пользователю, который их создал, и имена всех таблиц, принадлежащих данному пользователю, должны отличаться друг от друга, как и имена всех столбцов внутри данной таблицы. Порядок столбцов в таблице определяется порядком, в котором они указаны.

SQL позволяет создавать временные таблицы, «время жизни» которых — сеанс работы БД (время от открытия до закрытия базы данных).

Таблица может быть глобальной, т. е. доступной всей прикладной программе, создавшей ее:

```
CREATE GLOBAL TEMPORARY TABLE Продавцы
    (пном integer,
     пимя char (10),
     город char (10),
     комм decimal);
```

Она может быть локальной, доступной только для модуля программы, в котором она была создана:

```
CREATE LOCAL TEMPORARY TABLE Продавцы
    (пном integer,
     пимя char (10),
     город char (10),
     комм decimal);
```

Изменение таблицы после ее создания осуществляется командой ALTER TABLE. Обычно, она добавляет столбцы к таблице. Иногда она может удалять столбцы или изменять их размеры, а также в некоторых программах добавлять или удалять ограничения. Чтобы добавить столбец к таблице, используют формат:

```
ALTER TABLE <имя таблицы> ADD <имя поля>
<тип данных> <размер>;
```

Столбец будет добавлен последним и со значением NULL для всех строк таблицы.

Удаление проводится по команде

```
DROP TABLE <имя таблицы>;
```

Надо быть создателем таблицы, чтобы иметь возможность удалить ее.

Аналогично создаются и удаляются индексы.

Базовая таблица (5.5) представлена в простейшем виде и будет далее усовершенствована: другие таблицы будут созданы позднее. Сейчас рассмотрим лишь создание вида.

Структура и содержание видов. Только что созданная таблица называется базовой. Можно создавать представление (вид, View) — таблицы, содержимое которых берется или выводится из других таблиц. Вид создается командой CREATE VIEW:

```
CREATE VIEW Москва1
    AS SELECT *
    FROM Продавцы
    WHERE город = 'Москва';
```

Москва1 — представление (вид). Оно используется для целей защиты информации и вычисляется каждый раз при запросе и потому обновление данных автоматическое. Его можно использовать так же, как и любую другую базовую таблицу, однако в процедурах доступа и обновления имеется специфика.

Большое количество типов представлений доступно только для чтения. Это означает, что их можно запрашивать, но они не могут подвергаться действиям команд модификации.

Имеются некоторые виды запросов, которые не допустимы в определениях представлений: одиночное представление должно основываться на одиночном запросе; ОБЪЕДИНЕНИЕ (UNION) и ОБЪЕДИНЕНИЕ ВСЕГО (UNION ALL), агрегатные функции, DISTINCT в определении, вычисляемые поля не разрешаются в работе с представлениями; упорядочение (ORDER BY) никогда не используется в определении представлений.

Удаление представлений осуществляется (его владельцем) командой

```
DROP VIEW <имя вида>.
```

Заполнение БД данными. Значения могут быть помещены и удалены из полей командами языка DML (Язык Манипулирования Данными — ЯМД) INSERT (ВСТАВИТЬ), DELETE (УДАЛИТЬ) — выражения (5.2) и (5.3). Так, например, чтобы ввести строку в таблицу «Продавцы», можно использовать следующее условие:

```
INSERT INTO Продавцы
    VALUES (1001, 'Строков', 'Москва', .12);
```

Можно вставлять и пустое значение (NULL).

Возможно указывать столбцы таблицы в любом порядке, например,

```
INSERT INTO Заказчики (город, имя, номер)
    VALUES ('Москва', 'Иванов', 2001);
```

Отметим, что столбцы «рейтинг» и «пном» отсутствуют: эти строки автоматически установлены в значение NULL (по умолчанию).

Исключение строк, введенных по ошибке, проводится командой DELETE. Она может удалять только введенные строки, а не индивидуальные значения полей, так что параметр поля является необязательным или недоступным. Чтобы удалить все содержание таблицы «Продавцы», надо ввести:

```
DELETE FROM Продавцы;
```

Теперь таблица пуста и ее можно окончательно удалить командой DROP TABLE. Обычно нужно удалить только некоторые определенные строки из таблицы, для чего используется предикат. Например, чтобы удалить данные продавца Козлова из таблицы «Продавцы», можно ввести

```
DELETE FROM Продавцы
    WHERE пном = 1003;
```

Команды INSERT, DELETE совместно с командой UPDATE используются в процедуре обновления при эксплуатации БД.

Задание (обеспечение) целостности. Это разновидность команды CREATE TABLE, позволяющая устанавливать ограничения в таблицах.

Ограничение столбца вставляется в конец имени столбца после типа данных и перед запятой. Ограничения таблицы помещаются в конец имени таблицы после последнего имени столбца, но перед заключительной круглой скобкой:

```
CREATE TABLE <имя таблицы>
    (<имя столбца> <тип данных> <ограничение столбца>,
    (<имя столбца> <тип данных> <ограничение столбца>,
    <ограничение таблицы> ( <имя столбца>
    [, <имя столбца> ]...);
```

Перечислим некоторые ограничения.

1. Исключение пустых (NULL) указателей введением команды NOT NULL.

2. Уникальность данных и первичные ключи.

Ограничение столбца UNIQUE в поле при создании таблицы отклонит любую попытку ввода в это поле для одной из строк значения, которое уже представлено в другой строке. Это ограничение может применяться только к полям, которые были объявлены как непустые (NOT NULL).

3. SQL поддерживает первичные ключи непосредственно с ограничением PRIMARY KEY (первичный ключ). Первичные ключи не могут иметь значений NULL. Это означает, что, подобно полям в ограничении UNIQUE, любое поле, используемое в ограничении PRIMARY KEY, должно уже быть объявлено NOT NULL.

4. Ограничения на значения полей. Для этого используется ограничение CHECK: чтобы предотвратить ошибку неправильного введения значения «комм», наложим ограничение столбца — CHECK («комм» меньше, чем 1).

Сказанное может быть представлено в виде

```
CREATE TABLE Продавцы
    (пном integer NOT NULL PRIMARY KEY,
     пимя char(10) NOT NULL UNIQUE,
     город char(10),
     комм decimal CHECK (комм < 1));
```

Могут быть заданы интервалы и множества значений ограничений.

Создадим предварительно таблицу «Заказы»:

```
CREATE TABLE Заказы
    (прном integer NOT NULL UNIQUE,
     сумпр decimal,
     датпр date NOT NULL,
     зном integer NOT NULL,
     пном integer NOT NULL);
```

5. Установка значений по умолчанию.

Значение DEFAULT (ПО УМОЛЧАНИЮ) указывается в команде CREATE TABLE тем же способом что и ограничение столбца.

Если офис находится в Орле и подавляющее большинство продавцов тоже живут в Орле, то по умолчанию:

```
CREATE TABLE Продавцы
    (пном integer NOT NULL UNIQUE,
     пимя char(10) NOT NULL UNIQUE,
```

```
    город char(10) DEFAULT = 'Орел',
    комм decimal CHECK (комм < 1);
```

6. Ограничения на внешний ключ (ссылочная целостность). SQL поддерживает ссылочную целостность в команде CREATE TABLE (или ALTER TABLE) ограничением FOREIGN KEY с синтаксисом

```
FOREIGN KEY <список полей> REFERENCES
    <таблица с родительским ключом> [ <список полей>.
```

Создадим таблицу «Заказчики» с полем «пном», определенным в качестве внешнего ключа, ссылающегося на таблицу «Продавцы»:

```
CREATE TABLE Заказчики
    (знном integer NOT NULL PRIMARY KEY
     зимя char(10), город char(10),
     пном integer,
     FOREIGN KEY (пном) REFERENCES Продавцы(пном);
```

Ограничение могло быть внесено и отдельно:

```
CONSTRAINT поле_пном FOREIGN KEY (пном)
REFERENCES Продавцы(пном);
```

Такое введение ограничения удобно возможностью прямого удаления или записи другого выражения без изменения программы создания таблицы. Удаление ограничения

```
DROP CONSTRAINT поле_пном;
```

особенно удобно, если таблица заполнена.

Отметим, что родительский ключ должен быть уникальным и не содержать никаких пустых значений (NULL). Этого недостаточно для родительского ключа при объявлении внешнего ключа. SQL должен быть уверен что двойные значения или пустые значения (NULL) не были введены в родительский ключ, т. е.

```
CREATE TABLE Продавцы
(пном integer NOT NULL PRIMARY KEY,
пимя char(10) NOT NULL,
город char(10),
комм decimal);
```

```
CREATE TABLE Заказчики
(знom integer NOT NULL PRIMARY KEY,
зимя char(10) NOT NULL,
город char(10),
пном integer,
рейтинг integer
    FOREIGN KEY(пном) REFERENCES Продавцы,
    UNIQUE (знom, пном);

CREATE TABLE Заказы
(прном integer NOT NULL PRIMARY KEY,
сумпр decimal,
датпр date NOT NULL,
знom integer NOT NULL
пном integer NOT NULL
FOREIGN KEY (знom, пном) REFERENCES
Заказчики (знom, пном);
```

Таким образом, созданы все три таблицы и установлены связи между ними.

Если необходимо изменить или удалить текущее ссылочное значение родительского ключа, имеется три возможности:

1) ограничить или запретить изменение (способом ANSI), обозначив, что изменения в родительском ключе ограничены (RESTRICTED);

2) можно сделать изменение в родительском ключе и тем самым автоматические изменения во внешнем ключе, т. е. каскадное изменение (CASCADES);

3) провести изменение в родительском ключе и установить автоматически внешний ключ в NULL (полагая, что NULLS разрешен во внешнем ключе) — пустое изменение внешнего ключа (NULL).

Например, обновление и уничтожение

```
CREATE TABLE Заказчики
    (знom integer NOT NULL PRIMARY KEY,
     зимя char(10) NOT NULL,
     город char(10),
     рейтинг integer,
     пном integer REFERENCES Продавцы,
     UPDATE OF CASCADES,
     DELETE OF Продавцы RESTRICTED);
```

Если теперь попробовать удалить Строкова из таблицы «Продавцы», команда будет не допустима, пока не будет изменено значение поля «пном» «Заказчики» и Иванов и Крабов — для другого назначенного продавца.

В то же время можно изменить значение поля «пном» для Строкова на 1009, а у данных Иванов и Крабов значения поля «пном» будут также автоматически изменены.

Возможны и NULL (пустые) изменения. Например,

```
CREATE TABLE Заказы
пном integer NOT NULL PRIMARY KEY,
сумпр decimal,
датпр date NOT NULL,
зном integer NOT NULL REFERENCES Заказчики,
пном integer REFERENCES Продавцы,
    UPDATE OF Заказчики CASCADES,
    DELETE OF Заказчики CASCADES,
    UPDATE OF Продавцы CASCADES,
    DELETE OF Продавцы NULLS);
```

Естественно, что в команде DELETE с эффектом NULL в таблице «Продавцы» ограничение NOT NULL должно быть удалено из поля «пном».

С помощью языка SQL могут быть заданы база данных, хранимая процедура, генератор, счетчик, триггер.

База данных формируется так:

```
CREATE DATABASE «d:\...\a1.gdb»
    USER «SYSDBA»
    PASSWORD «masterkey»
    Default character set WIN1251
```

Хранимая процедура:

```
CREATE PROCEDURE Rashod_Tovara(IN_Tovar varchar (20))
    RETURNS(OUT_Tovar varchar(20))
    AS
    BEGIN
    FOR SELECT Tovar
        FROM Rashod
            WHERE Tovar=:IN_Tovar
    INTO :OUT_Tovar;
```

```
    DO
    SUSPEND;
END
```

Генератор задается двумя процедурами:

```
CREATE PROCEDURE Get_N_Rash;
        SET GENERATOR RASHOD_N_Rash TO 1;

CREATE PROCEDURE Get_N_Rash
        RETURNS(NR integer)
        AS
        BEGIN
        NR=Gen_ID(RASHOD_N_Rash, 1);
END;
```

Триггер каскадного обновления:

```
CREATE TRIGGER BU_Tovary
    ACTIVE
    BEFORE UPDATE
    AS
    BEGIN
    IF (OLD.Tovar<>NEW.Tovar) THEN
    UPDATE Rashod
    SET Tovar=NEW.Tovar
    WHERE Tovar=OLD.Tovar;
END
```

Система разрешений. Возможна система разрешения (привилегий) или запрета доступа к данным. Напомним, что администраторы баз данных сами создают пользователей и дают им привилегии. Однако и пользователи, которые создают таблицы, имеют права на управление этими таблицами.

Привилегии — это то, что определяет, может ли указанный пользователь выполнить данную команду.

Рассмотрим основные принципы их построения на базе языка SQL.

Различают следующие приоритеты (по убыванию): роль, пользователь, группа, общий доступ (public).

Имеются разрешающие и запрещающие действия.

Разрешение дается оператором вида

```
GRANT <вид операции>
        ON <объект>
    TO <субъект>
    [WITH GRANT OPTION].
```

Последняя строка говорит о передаче права пользования.

Для уверенного управления необходимо сочетание администратора БД и владельца объекта (например, таблицы).

Создание пользователя Илья (которому предоставляется возможность создания баз данных), выполняемое администратором БД, определяется командой

```
CREATE USER Илья
            WITH PRIVILEGES create_db;
```

лишение этого пользователя привилегий осуществляется командой

```
DROP USER Илья;
```

создание роли (с заданием пароля) проводится командой

```
CREATE ROLE create_db WITH PASSWORD = '12';
```

а лишение роли — командой

```
DROP ROLE create_db;
```

Имеется несколько типов привилегий, соответствующих нескольким типам операций. Привилегии даются и отменяются двумя командами SQL: GRANT (*допуск*) и REVOKE (*отмена*).

Каждый пользователь в среде SQL имеет специальное идентификационное имя (идентификатор — ID) или номер.

ID-разрешения — это имя пользователя. SQL может использовать специальное ключевое слово USER, которое ссылается на идентификатор доступа, связанный с текущей командой. Команда интерпретируется и разрешается (или запрещается).

Привилегии объекта связаны одновременно и с пользователями, и с таблицами. Следует помнить, что пользователь, создавший таблицу (любого вида), является владельцем этой таблицы: он имеет все привилегии в этой таблице и может передавать эти привилегии другим пользователям.

Привилегии относятся к командам SELECT, INSERT, UPDATE, DELETE, REFERENCES (определение внешнего ключа, который использует один или более столбцов этой таблицы, как родительский ключ).

К нестандартным командам относятся INDEX (индекс), дающий право создавать индекс в таблице, и ALTER (изменить) — для выполнения команды ALTER TABLE в таблице. Механизм SQL назначает пользователям эти привилегии с помощью команды GRANT.

Например, пользователь Илья имеет таблицу «Заказчики» и хочет позволить пользователю Петр выполнить запрос к ней:

```
GRANT SELECT ON Заказчики TO Петр;
```

Петр может выполнить запросы к таблице «Заказчики», но не может предоставить право SELECT другому пользователю: таблица еще принадлежит Илье.

Возможны и групповые привилегии:

```
GRANT SELECT, INSERT ON Заказы TO Илья, Петр;
```

Для команд UPDATE и REFERNCES можно указывать и отдельные поля:

```
GRANT UPDATE (комм) ON Продавцы TO Илья;
```

Для привилегии REFERENCES может быть указан список из одного или более столбцов, для которых ограничена эта привилегия. Например, Илья может предоставить Степану право использовать поля таблицы «Заказчики», как таблицу родительского ключа, с помощью такой команды:

```
GRANT REFERENCES (зимя, зном)
        ON Заказчики TO Степан;
```

SQL поддерживает два аргумента для команды GRANT, которые имеют специальное значение: ALL PRIVILEGES (все привилегии) или просто ALL — для команд и PUBLIC (общие) — для пользователей.

```
GRANT ALL ON Заказчики TO PUBLIC;
```

Возможность передачи пользователем предоставленных ему привилегий осуществляется предложением WITH GRANT OPTION.

Иногда создателю таблицы хочется, чтобы другие пользователи могли получить привилегии в его таблице. Обычно это делается в системах, где один или более людей создают несколько (или все) базовые таблицы в базе данных, а затем передают ответственность за них тем, кто будет фактически с ними работать. SQL позволяет делать это с помощью предложения WITH GRANT OPTION.

Пусть Илья передает право Петру на привилегию SELECT в таблице «Заказчики»:

```
GRANT SELECT ON Заказчики TO Петр
    WITH GRANT OPTION;
```

Возможно разрешить выполнение процедуры integ_check

```
GRANT EXECUTE
    ON PROCEDURE integ_check
        TO PUBLIC;
```

Привилегия, например, на команду INSERT, может быть отменена:

```
REVOKE INSERT ON Заказы FROM Петр;
```

Здесь также можно использовать списки:

```
REVOKE INSERT, DELETE ON Заказчики
    FROM Петр, Степан;
```

Возможно задание (или отмена) привилегий с помощью рассмотренных ранее видов. Например,

```
CREATE VIEW Москва1
    AS
    SELECT зимя, пимя
        FROM Продавцы;
```

предоставляет Москва1 привилегию SELECT в виде (представлении), а не в самой таблице «Продавцы»:

```
GRANT SELECT ON Москва1 TO Виктор;
```

Привилегии возможно ограничивать и строками:

```
CREATE VIEW Москва2
    AS SELECT *
        FROM Заказчики
        WHERE город = 'Сочи'
        WITH CHECK OPTION;
    GRANT UPDATE ON Москва2 TO Петр;
```

Предложение WITH CHECK OPTION предохраняет Петр от замены значения поля «город» на любое значение, кроме Сочи.

Существует целый ряд вариантов работы с видами [26].

В системе любого размера всегда имеются некоторые типы суперпользователей — чаще всего Администратор Базы Данных или DBA. У него есть такие системные привилегии: CONNECT (подключить), RESOURCE (ресурс) и DBA (Администратор Базы Данных).

CONNECT состоит из права зарегистрироваться и права создавать представления и синонимы, RESOURCE состоит из права создавать базовые таблицы, DBA — это привилегия, дающая пользователю высокие полномочия в базе данных.

Некоторые системы имеют специального пользователя, иногда называемого SYSADM или SYS (Системный Администратор Базы Данных), который имеет наивысшие полномочия.

Команда GRANT, в измененной форме, является пригодной для использования с привилегиями объекта, как и с системными привилегиями:

```
GRANT RESOURCE TO Мирон;
```

Естественно, пользователь Мирон должен быть создан.

У пользователя может быть и пароль (например, Иван). Тогда команда имеет вид

```
GRANT CONNECT TO Федор IDENTIFIED BY Иван;
```

что приведет к созданию пользователя с именем Федор, даст ему право регистрироваться и назначит ему пароль Иван.

Если нужно запретить пользователю регистрироваться, следует использовать для REVOKE привилегию CONNECT, которая «удаляет» этого пользователя.

Запрещение (на создание таблиц в БД newa группе clerck) имеет вид

```
GRANT NOCREATE
    ON DATABASE newa
        TO GROUP clerk;
```

Многопользовательский режим. До сих пор подразумевался однопользовательский режим работы БД. SQL позволяет реализовать и многопользовательский режим, в том числе размещение запоминаемых данных, восстановление и сохранение изменений в базе данных, конфигурации вашей базы данных.

Команда или группа команд SQL могут быть выполнены или полностью проигнорированы с помощью транзакций. Транзакция начинается всякий раз, когда начинается сеанс с SQL. Все команды, которые введены, будут частью этой транзакции, пока они не завершатся вводом команды COMMIT WORK или команды ROLLBACK WORK. COMMIT может сделать все изменения постоянными с помощью транзакции, а ROLLBACK может откатить их обратно или отменить. Новая транзакция начинается после каждой команды COMMIT или ROLLBACK.

Синтаксис команд соответственно COMMIT WORK; или ROLLBACK WORK;

В большинстве реализаций можно установить параметр, называемый AUTOCOMMIT. Он автоматически запоминает все действия, которые выполняются. Действия, которые приведут к ошибке, всегда будут автоматически «прокручены» обратно. Для фиксации всех действий можно использовать команду:

```
SET AUTOCOMMIT ON;
```

Вернуться к обычной диалоговой обработке запросов можно с помощью такой команды:

```
SET AUTOCOMMIT OFF;
```

Имеется возможность установки AUTOCOMMIT, которую система выполнит автоматически при регистрации.

Обработка одновременных транзакций называется *параллелизмом* и может приводить к конфликтам.

SQL имеет средства управления параллелизмом для точного указания места получения результата: ни одна команда не должна быть выдана, пока предыдущая не будет завершена (включая команды COMMIT или ROLLBACK).

Механизм, используемый SQL для управления параллелизмом операций, называется блокировкой. Блокировки задерживают определенные операции в базе данных. Задержанные операции выстраиваются в очередь и выполняются только после снятия блокировки.

Существует несколько базовых типов блокировок: распределяемые, специальные и общие блокировки.

Распределяемые (или S-блокировки) могут быть установлены более чем одним пользователем в данный момент времени. Это дает возможность любому числу пользователей обращаться к данным, но не изменять их.

Специальные блокировки (или X-блокировки) не позволяют никому, кроме владельца этой блокировки, обращаться к данным. Специальные блокировки используются для команд которые изменяют содержание или структуру таблицы. Они действуют до конца транзакции.

Общие блокировки используются для запросов. Насколько они продолжительны — зависит фактически от уровня изоляции (сколько таблиц будет блокировано).

Использование БД

Обновление данных. К командам обновления относят INSERT, DELETE, рассмотренные при заполнении БД данными, и UPDATE, изменяющей некоторые или все значения в существующей строке. В команде названо имя используемой таблицы и предложение SET, указывающее на изменение, которое нужно сделать для определенного столбца. Например, чтобы изменить оценки всех заказчиков на 200, можно ввести

```
UPDATE Заказчики
    SET рейтинг = 200;
```

UPDATE, наподобие DELETE, может работать с предикатами. Например, можно выполнить изменение, одинаковое для всех заказчиков продавца Строкова (имеющего «пном» = 1001):

```
UPDATE Заказчики
SET рейтинг = 200
WHERE пном = 1001;
```

Можно изменить и несколько столбцов. Допустим, продавец Аврорин уволился, и надо переназначить его номер новому продавцу:

```
UPDATE Продавцы
    SET пимя = 'Суворов', город = 'Клин', комм = .10
    WHERE пном = 1004;
```

Команда REFERENCES может изменять значения в уже вставленных строках.

Запрос — команда, которая с помощью программы базы данных выводит определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминал.

Запрос формируется командой SELECT (выражение (5.4)).

В самой простой форме команда SELECT просто извлекает информацию из таблицы. Например, вывод всей таблицы «Продавцы» (см. табл. 5.3)

```
SELECT пном, пимя, город, комм
    FROM Продавцы;                                    (5.6)
```

Точка с запятой (;) используется во всех интерактивных командах SQL, чтобы сообщать базе данных, что команда закончена и готова выполниться. Звездочка (*) в команде может применяться для вывода полного списка столбцов и предыдущая процедура может выполняться командой:

```
SELECT *
FROM Продавцы;
```

Имеется большое число вариантов команды SELECT [26], из которых, в силу ограниченного объема данной работы, рассмотрим основные.

В выражении (5.6) можно задать не все поля или переупорядочить поля.

Предложение WHERE команды SELECT позволяет устанавливать предикаты, условие которых может быть или верным или неверным для любой строки таблицы. Команда извлекает только те строки из таблицы, для которых такое утверждение верно:

```
SELECT пимя, город
    FROM Продавцы
    WHERE город = 'Москва';
```

В предложении WHERE используются не только сравнения вида <, >, <>, =, ≤, ≥, но и области (множества), определяемые выражениями IN, BETWEEN, LIKE, IS (NOT) NULL.

К агрегатным функциям относят COUNT (количество строк в таблице), SUM (арифметическую сумму всех выбранных значений данного поля), AVG (среднее всех выбранных значений данного поля), MAX или MIN (наибольшее или наименьшее из всех выбранных значений данного поля).

Вывод агрегатных функций (вычисляемых полей) имеет такой вид:

```
SELECT SUM (сумпр)
    FROM Заказы;
```

Он определяет в данном случае сумму всех покупок в таблице «Заказы». Аналогична команда для остальных функций.

Предложение GROUP BY команды SELECT позволяет определять подмножество значений в особом поле в терминах другого поля, и применять функцию агрегата к подмножеству. Это дает возможность объединять поля и агрегатные функции в едином предложении SELECT. Например, найти наибольшую сумму приобретений, полученную каждым продавцом:

```
SELECT пном, MAX (сумпр)
    FROM Заказы
    GROUP BY пном;
```

Получим

```
пном
1001   767.19
1002  1713.23
1003    75.75
1014  1309.95
1007  1098.16
```

GROUP BY применяет агрегатные функции независимо от серий групп, которые определяются с помощью значения поля в целом.

Предложение HAVING определяет критерии, необходимые для удаления определенных групп из вывода. Тогда команда имеет вид

```
SELECT пном, датпр, MAX ((сумпр))
    FROM Заказы
    GROUP BY пном, датпр
    HAVING MAX ((сумпр)) > 3000.00;
```

Аргументы в предложении HAVING следуют тем же самым правилам, что и в предложении SELECT, состоящем из команд, использующихся в GROUP BY. Они должны иметь одно значение на группу вывода. В строгой интерпретации ANSI SQL нельзя использовать агрегат агрегата.

Если надо выполнить простые числовые вычисления данных (вычисляемые выражения) и затем поместить их в форму, более соответствующую потребностям, то SQL позволяет это сделать. Например, если ввести команду

```
SELECT пном, пимя, город, комм *100
    FROM Продавцы;
```

то в поле «комм» появятся значения в процентах.

Возможно упорядочение вывода по одному или нескольким полям, например, по убыванию:

```
SELECT *
    FROM ЗАКАЗЫ
    ORDER BY знom DESC, сумпр DESC;
```

Вместо имен столбца можно использовать их порядковые номера для указания поля, используемого в упорядочении вывода. Эти номера могут ссылаться не на порядок столбцов в таблице, а на их порядок в выводе: поле, упомянутое в предложении SELECT первым, для ORDER BY не зависит от того, каким по порядку оно стоит в таблице. Например, команда упорядочения по убыванию значения комиссионных такова:

```
SELECT пимя, комм
    FROM Продавцы
    GROUP BY 2 DESC;
```

пном	комм
Строков	0.17
Кирюшин	0.13
Удалов	0.15

До сих пор речь шла о работе с одной таблицей, однако возможна работа и с несколькими таблицами. Формат команд меняется мало, при этом можно использовать практически все, что применялось для одиночных таблиц.

Создание объединения чаще всего осуществляется командой вида

```
SELECT Заказчики.зимя, Продавцы.пимя,
    Продавцы.город
        FROM Продавцы, Заказчики
        WHERE Продавцы.город = Заказчики.город;
```

Здесь после SELECT указываются через точку имя таблицы и файла:

зимя	пимя	город
Иванов	Строков	Москва
Иванов	Строков	Москва
Ковров	Кирюшин	Пермь
Конкин	Кирюшин	Пермь
Иванов	Аврорин	Москва
Крабов	Аврорин	Москва

В указании полей могут использоваться и другие условия, например,

```
SELECT пимя, зимя
    FROM Продавцы, Заказчики
    WHERE пимя < зимя
    AND рейтинг < 200;
```

пимя	зимя
Строков	Красин
Аврорин	Красин
Козлов	Иванов
Козлов	Крабов
Козлов	Красин

Аналогично выглядит и команда соединения трех и более таблиц.

Заметим, что возможно соединение таблицы самой с собой. Эта схема может использоваться для проверки правильности заполнения таблицы. Для этого используются синонимы таблиц (алиасы). Например, найти все пары заказчиков, имеющих одинаковый рейтинг:

```
SELECT перв.зимя, втор.зимя, перв.рейтинг
FROM Заказчики перв, Заказчики втор
WHERE перв.рейтинг = перв.рейтинг;
        Петров      Петров      200
        Петров      Ковров      200
        Ковров      Петров      200
        Ковров      Ковров      200
        Квакин      Квакин      300
        Квакин      Конкин      300
        Крабов      Иванов      100
        Крабов      Крабов      100
        Крабов      Красин      100
        Конкин      Квакин      300
        Конкин      Конкин      300
        Красин      Иванов      100
        Красин      Крабов      100
        Красин      Красин      100
```

В вышеупомянутой команде SQL ведет себя так, как если бы он соединял две таблицы, называемые «первая» и «вторая». Обе они — фактически таблицы «Заказчики», но псевдонимы разрешают им быть обработанными независимо. Псевдонимы «первый» и «второй» были установлены в предложении FROM запроса, сразу после имени копии таблицы. Псевдонимы могут использоваться в предложении SELECT, даже если они не определены в предложении FROM.

Следующая команда будет определять любые ошибки в заполнении:

```
SELECT перв.прном, втор.зном, перв.пном,
    втор.прном, втор.зном, втор.пном
    FROM Заказчики перв, Заказчики втор
    WHERE перв.зном = втор.зном
    AND перв.пном < > втор.пном;
```

Если заполнение таблицы проведено верно, то вывод будет пустым.

При работе с несколькими таблицами запрос может быть составным, использующим или не использующим операции обновления. Внутренние запросы называют подзапросами.

Пусть известно имя продавца (Аврорин), но значение его поля «пном» неизвестно и надо извлечь все заказы из таблицы «Заказы». Тогда:

```
SELECT *
    FROM Заказы
    WHERE пном =
    (SELECT пном
        FROM Продавцы
        WHERE пимя = 'Москва');
```

Чтобы оценить внешний (основной) запрос, SQL сначала должен оценить внутренний запрос (подзапрос) предложения WHERE. Ответ: «пном» = 1004. Однако SQL, не просто выдает это значение, а помещает его в предикат основного запроса вместо самого подзапроса, так чтобы предикат прочитал

```
WHERE пном = 1004.
```

Для работы с несколькими таблицами может использоваться и операция объединения (UNION). Например, для получения всех продавцов и заказчиков, размещенных в Москве можно использовать команду:

```
SELECT пном, пимя
    FROM Продавцы
    WHERE город = 'Москва'
    UNION
    SELECT зном, зимя
    FROM Заказчики
    WHERE город = 'Москва';
```

Как можно видеть, столбцы, выбранные двумя командами, выведены так, как если она была одна. Заголовки столбца исключены, потому что ни один из столбцов, выведенных объединением, не был извлечен непосредственно только из одной таблицы. Только последний запрос заканчивается точкой с запятой. Отсутствие точки с запятой дает понять SQL, что имеется еще один или более запросов.

Возможны и запросы с командами обновления, рассмотренными ранее.

Можно выполнить и так называемый перекрестный запрос, результатом которого является таблица, аналогичная таблице в приложении Excel. Команда TRANSFORM служит для выполнения такого запроса.

До сих пор речь шла о *статическом языке SQL*. Дадим характеристику *динамическому языку SQL*. Первую часть предыдущего сложного оператора можно записать

```
SELECT пном, пимя
    FROM Продавцы
    WHERE город = :город;
```

Здесь двоеточие перед словом «город» означает, что задан параметр. Его значение либо вводит пользователь в процессе работы, либо оно заимствуется из других таблиц в процессе выполнения программы. Параметры могут быть заданы во всех командах выборки и обновления.

♦ ВСТРОЕННЫЙ ЯЗЫК SQL

До сих пор рассматривался интерактивный SQL. Теперь кратко остановимся на встроенном языке SQL, используемом для расширения программ, написанных на других языках. Хотя непроцедурность интерактивного языка SQL делает его очень мощным, в то же время она накладывает на язык большое число ограничений.

Чтобы преодолеть эти ограничения, можно включать SQL в программы, написанные на каком-либо процедурном языке. В качестве такого процедурного языка примем Паскаль, поскольку он прост в понимании и имеет полуофициальный стандарт ANSI.

Отметим некоторые ограничения SQL.

Логические конструкции типа if ... then (если ... то), for ... do (для ... выполнить) и while ... repeat (пока ... повторять), используемые для структур большинства компьютерных программ, здесь отсутствуют.

Интерактивный SQL не может делать многие операции со значениями, размещением или распределением их, выводом их на какое-то устройство. Цель встроенного SQL состоит в соединении возможностей декларативного и процедурного языков.

Строго говоря, следует выделить вложенный и встроенный язык SQL.

В первом случае основным является язык SQL, в который вводятся операторы циклов и условных переходов. Здесь нет резкого разделения на интерфейсный и вложенный языки. Обычно операторы выборки (SELECT), обновления (DELETE, UPDATE, INSERT), перекрестного запроса (TRANSFORM) относят к интерактивному языку SQL. В то же время операторы CREATE, ALTER, DROP,

GRANT, REVOKE считаются принадлежащими к вложенному языку SQL. В качестве объектов выступают таблицы, поля, ограничения, индексы, домены, виды, генераторы, триггеры, хранимые процедуры. Присутствие «алгоритмических составляющих» языка лучше всего видно в СУБД InterBase в триггерах и хранимых процедурах.

Во втором случае операторы языка SQL «встраиваются» в алгоритмические языки. Часто им является язык Pascal. Для этого команды SQL помещаются в исходный текст главной программы, которой предшествует фраза EXEC SQL (EXECute SQL). Далее обсуждаются некоторые команды, являющиеся специальными для вложенной формы SQL.

Когда вставляются команды SQL в текст программы, написанной на другом языке, надо выполнить прекомпиляцию прежде, чем окончательно ее скомпилировать. Программа, называемая прекомпилятором (или препроцессором), будет просматривать текст программы и преобразовывать команды SQL в форму, удобную для использования базовым языком. Затем используется обычный транслятор, чтобы преобразовывать программу из исходного текста в выполняемый код.

Иными словами, основная программа вызывает процедуры SQL, которые выбирают параметры из главной программы и возвращают уже обработанные значения в основную программу. Модуль может содержать любое число процедур, каждая из которых состоит из одиночной команды SQL. Идея в том, чтобы процедуры могли работать тем же самым способом, что и процедуры на языке, в который они были вложены.

Каждая из программ, использующих встроенный SQL, связана с идентификатором (ID) доступа во время ее выполнения. Идентификатор доступа, связанный с программой, должен иметь все привилегии, чтобы выполнять операции SQL, выполняемые в программе.

SQL и части базового языка программ будут связываться друг с другом с помощью значений переменных. Разные языки распознают различные типы данных для переменных и ANSI определяет эквиваленты SQL базового языка Паскаль.

Можно использовать переменные из главной программы во вложенных операторах SQL везде, где будут использоваться выражения значений.

Главные переменные должны:

- быть объявленными в SQL DECLARE SESSION (раздел объявлений);
- иметь совместимый тип данных с их функциями в команде SQL;

• быть назначенными значению во время их использования в команде SQL, если команда SQL самостоятельно не может сделать назначение;

• предшествовать двоеточию (:), когда они упоминаются в команде SQL.

Пусть в программе имеются четыре переменных с именами id_num, salesperson, loc и comm. Они содержат значения, которые нужно вставить в таблицу «Продавцы». Можно вложить следующую команду SQL в программу

```
EXEC SQL INSERT INTO Продавцы
    VALUES (:id_num, :salesperson, :loc, :comm)
```

Обратите внимание, что точка с запятой в конце команды отсутствует. Это потому, что соответствующее завершение для встроенной команды SQL зависит от языка, для которого делается вложение.

Команду можно включить в цикл:

```
while not end-of-file (input) do
begin
readln (id_num, salesperson, loc, comm);
EXEC SOL INSERT INTO Salespeople
    VALUES (:id_num, :salesperson, :loc, :comm);
end;
```

Фрагмент программы на языке Паскаль определяет цикл, который будет считывать значения из файла, сохранять их в четырех поименованных переменных, сохранять значения этих переменных в таблице «Продавцы» и затем считывать следующие четыре значения.

Все переменные, на которые имеется ссылка в предложениях SQL, должны сначала быть объявлены в SQL DECLARE SECTION (раздел объявлений), использующем обычный синтаксис главного языка. Раздел объявлений должен начинаться и кончаться встроенными командами SQL — BEGIN DECLARE SECTION (начало раздела объявлений) и END DECLARE SECTION (конец раздела объявлений), которым предшествует, как обычно EXEC SQL (выполнить).

Чтобы объявить переменные предыдущего примера, можно ввести:

```
EXEC SQL BEGIN DECLARE SECTION;
Var
```

```
id-num:          integer;
Salesperson:     packed array (1 .. 10) of char;
loc:         packed array (1 .. 10) of char;
comm:            real;
EXEC SQL END DECLARE SECTION;
```

где Var — заголовок, который предшествует ряду объявляемых переменных и упакованным (или распакованным) массивам.

Переместим строку Строков из таблицы «Продавцы» в переменные главного языка:

```
EXEC SQL SELECT snum, sname, city, comm
    INTO :id_num, :salesperson, :loc, :comm
    FROM Salespeople
    WHERE snum = 1001;
```

Возможно использовать запросы, выводящие многочисленные строки, применяя курсор. Курсор — это вид переменной, которая связана с запросом. Значением этой переменной может быть каждая строка, которая выводится при запросе.

Заметим, что язык SQL может использоваться как для построения, так и для запросов, в то время как язык QBE применяется только для обновления и, в основном, для запросов.

Другим вариантом встроенного языка программирования SQL служит язык, используемый, например, при применении СУБД InterBase в среде Delphi: так называемый формируемый запрос, в котором SQL-операторы встраиваются в язык Object Pascal с помощью команды Query1.SQL.Add('...'), где в скобках указывается SQL-оператор.

ЯЗЫК QBE

Работа на языке SQL проста, но требует знания языка. Для пользователей, совершенно не знакомых с программированием, запрос может осуществляться одним из следующих способов:

1) через меню и экранные формы;

2) с помощью языка Query By Example — QBE (запрос по примеру).

Первый способ зависит от варианта разработки СУБД, второй способ, предложенный М.М. Злудом и базирующийся на *исчислении доменов*, является универсальным и не требует знания программирования. В связи с этим рассмотрим подробнее идею построения языка.

С помощью этого способа на экран вызывается одна или несколько таблиц. В первом столбце каждой из них указывается имя таблицы (файла). В столбцах, под именами полей, могут быть указаны условия запроса. Переход от одной таблицы к другой осуществляется с помощью указания в соответствующих полях одинаковых подчеркнутых переменных, написанных большими буквами. Выводимые на печать поля указываются символом P (print), точкой и подчеркнутой переменной (например, P.A).

Приведем два примера запросов, базирующихся на примере 4.1.

Пример 5.2. Получить фамилии студентов 4 курса кафедры ИиУС, имеющих балл не менее 4 (рис. 5.3).

Файл	Номер студента	Фамилия студента	Балл	Курс	Группа	Кафедра
СТУД		P.A	>=4	4		

Рис. 5.3. Запрос фамилии по одной таблице

Пример 5.3. Получить фамилии студентов, руководителем которых является профессор кафедры ИиУС (рис. 5.4).

Файл	Номер студента	Фамилия студента	Балл	Курс	Группа	Кафедра
СТУД	X	P.A				

Файл	Номер преподавателя	Фамилия преподавателя	Должность	Кафедра	Зарплата	Надбавка
ПРЕП	Y		проф	ИиУС		

Файл	Номер студента	Номер преподавателя	Часовая нагрузка
СТПР	X	Y	

Рис. 5.4. Запрос фамилии по нескольким таблицам

Могут быть использованы (в первом столбце запроса) агрегатные функции (SUM, AVG, MAX, MIN, CNT-счетчик). Возможно создавать виды (View).

QBE поддерживает справочник таблиц [4], процедуры обновления.

Реализация языка программирования QBE может иметь несколько вариантов. Один из них для СУБД Access рассмотрен в гл. 5.

Контрольные вопросы

1. Что такое «отношение»?
2. Назовите характеристики отношения.
3. Что такое арность отношения? размерность? ключ?
4. Для чего используются ключи?
5. Что такое составной ключ (суперключ)? родительский и внешний ключи?
6. В чем цель нормализации?
7. Сформулируйте назначение 1—5 нормальных форм.

Глава 6

Сетевые и иерархические базы данных

При изложении сетевой и иерархической моделей данных целесообразно использовать такой порядок.

Сначала описывается логическая структура модели данных (МД), затем — процессы использования построенной БД с помощью предложенного Э.Ф. Коддом языка Альфа [9] и, наконец, процедура программного описания (создания и использования) БД. Поскольку Альфа-описание указанных моделей данных не отличается от Альфа-описания реляционной модели данных, оно опущено.

6.1. ЛОГИЧЕСКАЯ СТРУКТУРА СЕТЕВОЙ БД

При различных способах реализации сетевых моделей [1—11] наибольшее распространение получила модель КОДАСИЛ (CODASYL COnference on DAta SYstems Language — Ассоциация по языкам систем обработки данных), предложенная Рабочей группой по базам данных (DBTG — DataBase Task Group). Эта модель считается наиболее развитой сетевой моделью данных, постоянно развивается, поддерживается и сопровождается, являясь как бы стандартом.

Ассоциация КОДАСИЛ образовалась в 1959 г., а Рабочая группа БД, начиная с 1969 г., выпускала спецификации. Основы сетевой модели были заложены в 1971 г. Комитетом по ЯОД. Вместо Рабочей группы БД стала работать Рабочая группа языков БД, начавшая с расширения КОБОЛа. Серьезные результаты были получены в 1973 г. В отчетах группы была описана сетевая модель данных, фактически мало изменившаяся с тех пор. Основная цель КОДАСИЛ — создание иерархической модели, позволяющей описывать отношения M : M, т. е. уменьшить недостатки иерархической модели.

Разрабатывались и соответствующие СУБД: DMS (корпорации UNIVAC), IDMS (Cullinane), DBMS (DEC), IDS (Honeywell). В настоящее время широко известна db_Vista, работающая на персональных компьютерах в DOS и Windows.

Структурными элементами сетевой модели данных КОДАСИЛ являются элемент данных, агрегат данных и запись (рис. 6.1, *а*), которым присваиваются имена.

Элемент данных (ЭД) — наименьшая поименованная единица данных (аналог поля в файловых системах), с помощью которой осуществляется построение всех остальных структур. Примеры элементов данных: «Табельный номер», «Шифр детали», «Год рождения». Имя элемента данных используется для его идентификации в схеме структуры данных более высокого уровня. Значение элемента данных может быть числовым (целым, вещественным), нечисловым (символьным, логическим). В некоторых приложениях может использоваться «неопределенное» значение элемента данных и это значит, что значение соответствующего свойства объекта не определено в БД.

Агрегат данных — поименованная совокупность элементов данных внутри записи, которую можно рассматривать как единое целое. Имя агрегата используется для его идентификации в схеме структуры данного более высокого уровня. Агрегат данных может быть простым (рис. 6.1, *б*), если состоит только из элементов данных, и составным (рис. 6.1, *в*), если включает в свой состав другие агрегаты.

а)

б)

Дата		
Число	Месяц	Год

в)

Предприятие			
Название	Адрес		
	Почтовый индекс	Город	Улица номер дома

г)

Заказ на покупку														
Номер заказа	Дата заказа			Партия товара										
	Число	Месяц	Год	Шифр товара	Кол-во	Цена	Шифр товара	Кол-во	Цена	...	Шифр товара	Кол-во	Цена

Рис. 6.1. Структурные элементы сетевой БД

Различают агрегаты типа «вектор» и типа «повторяющаяся группа». Агрегат, повторяющаяся компонента которого является простым элементом данных, называется «вектором». Например агрегат «Заработная плата», в котором экземпляр элемента данных может повторяться до 12 раз (за каждый месяц года). Агрегат, повторяющаяся компонента которого представлена совокупностью данных, называется повторяющейся группой. В повторяющуюся группу могут входить отдельные элементы данных, векторы, агрегаты или повторяющиеся группы. На рис. 6.1, *г* представлен агрегат «Заказ на покупку», имеющий в своем составе повторяющуюся группу «Партия товара».

Максимальное количество экземпляров для вектора и повторяющейся группы ограничено и задается при спецификации схемы записи.

Запись — поименованная совокупность элементов данных и/или агрегатов. Иными словами, запись — это агрегат, который не входит в состав никакого другого агрегата и может иметь сложную иерархическую структуру, поскольку допускается многократное применение агрегации. Имя записи используется для идентификации типа записи в схемах типов структур более высокого уровня.

Записи являются фактически простейшими структурными элементами, через которые осуществляются связи: элементом связи служит набор.

Набор записей — поименованная совокупность записей, образующая двухуровневую иерархическую структуру подчинения: каждый тип набора представляет собой отношение (связь) между двумя или несколькими типами записей. В силу иерархичности набора для каждого его типа один тип записи объявляется «владельцем», тогда остальные типы записей — «члены», т. е. имеют место «запись-владелец» и «запись-член (набора)». Каждый экземпляр набора должен содержать только один экземпляр записи-владельца и любое количество экземпляров записей-членов.

Структуры БД строятся на основании следующих основных композиционных правил.

1. База данных может содержать любое количество типов записей и типов наборов.

2. Между двумя типами записи может быть определено любое количество типов наборов.

3. Тип записи может быть владельцем и одновременно членом нескольких типов наборов.

Допускается добавление новой записи в качестве экземпляра владельца, если запись-член отсутствует.

При удалении записи-владельца удаляются соответствующие указатели на записи-члены, но сами записи-члены не уничтожаются (сингулярный набор).

Наборы могут быть нескольких разновидностей.

1. С одними и теми же типами записей, но разными типами наборов.

Рис. 6.2. Сингулярный набор

2. Наборы из трех и более записей, в том числе с обратной связью.

3. Сингулярный набор (только один экземпляр). У такого набора нет естественного владельца и в качестве его выступает система (рис. 6.2). В дальнейшем такие наборы могут приобрести запись-владельца.

Количество объявляемых сингулярных наборов произвольно. Один и тот же тип записи может быть объявлен членом сингулярного набора и одновременно владельцем либо членом других наборов.

Обычно тип набора задается между двумя типами записей. Однако в модели можно представлять типы связей, заданных между несколькими типами сущностей. Для этого используют многочленные наборы, которые представляют собой отношение между тремя или более типами записей, один из которых назначается владельцем набора, а остальные — членами набора. На рис. 6.3 показан пример многочленного набора «Научные труды», владельцем которого является запись типа «Научный сотрудник», а членами — записи типа «Научный отчет», «Доклад на конференции», «Статья в журнале»,

Рис. 6.3. Многочленный набор

«Монография». Экземпляр некоторого типа многочленного набора включает в себя один экземпляр записи-владельца и все связанные с ним экземпляры записей-членов заданных типов. В конкретных СУБД концепция многочленного набора может быть не реализована.

В сетевой БД возможны следующие способы доступа:

- последовательный просмотр записей основного файла;
- просмотр всех записей зависимого файла, связанного с конкретной записью основного файла;
- прямой поиск записи основного файла по ее ключу (точка доступа).

Отметим, что если запись используется для представления сущности, то набор — для представления связей между рассматриваемыми сущностями.

Сетевые модели данных могут быть представлены в форме графов: вершины графа используются для интерпретации типов сущностей, а дуги — для интерпретации типов связей между сущностями.

На графической диаграмме схемы БД тип набора изображается поименованной дугой между соответствующими типами записей: дуга исходит из типа записи-владельца набора и заходит в тип записи-члена набора.

В модели КОДАСИЛ основным внутренним ограничением целостности является функциональность связей, т. е. с помощью наборов можно реализовать непосредственно связи типа 1 : 1, 1 : M, M : 1. В модели это первое внутреннее ограничение выражается утверждением: в конкретном экземпляре набора экземпляр записи-члена набора может иметь не более одного экземпляра записи-владельца набора. Следовательно, число экземпляров набора некоторого типа в точности равно числу экземпляров записей-владельцев этого типа набора в БД. При этом экземпляр набора может быть и пустым, т. е. состоять из экземпляра записи-владельца (экземпляры записей-членов могут отсутствовать в некоторые моменты времени при функционировании системы).

Из функционального характера реализуемых в МД связей следует второе внутреннее ограничение целостности — экземпляр записи может быть членом только одного экземпляра среди всех экземпляров набора одного типа (он может входить в состав двух и более экземпляров наборов, но различных типов).

Функциональный характер реализуемых связей не позволяет непосредственно представлять в модели тип «многие ко многим». Для представления связи типа M : N вводят вспомогательный тип записи и две функциональные связи типа 1 : M (см. рис. 4.7).

6.2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СЕТЕВОЙ БД

Представим в виде сетевой модели БД «Учебный процесс». Возьмем за основу связи, показанные на рис. 3.4, *б* (см.). Здесь по-прежнему присутствуют связи M:N. Их замена возможна двумя способами:

- использованием нормализации (рис. 3.4, *в*);
- введением, наряду с прямыми, обратных связей.

Второй путь характерен для иерархических МД. Поскольку в сетевой МД используются и ключи, ей более присущ первый путь. Тогда сетевая модель получит вид, показанный на рис. 6.4, а некоторые экземпляры логических записей — на рис. 6.5.

Отметим, что в отчетах КОДАСИЛ первоначально предполагалось обязательное участие программиста в работе БД. В последующих версиях предусматривалось минимальное участие программиста и потому ряд команд был аннулирован.

В сетевой модели данных нет полной независимости логической БД от физической БД. Это хорошо заметно из рис. 6.5 и программы на языке СУБД DBMS, приведенной ниже.

```
SCHEMA NAME IS DB_2

AREA NAME IS DB2_AREA

RECORDNAME IS PART   .
LOCATION MODE IS CALC HASH_PS
USING PS in PART
DUPLICATES NOT ALLOWED
WITHIN DB2_AREA
PS TYPE IS CHAR 5
PD TYPE IS CHAR 25
CL TYPE IS CHAR 2

RECORDNAME IS WH
WITHIN DB2_AREA
WS TYPE IS CHAR 5
WD TYPE IS CHAR 25

SET NAME IS INVENTIORY
OWNER IS PART
MEMBER IS WH
```

Рис. 6.4. Структура сетевой БД

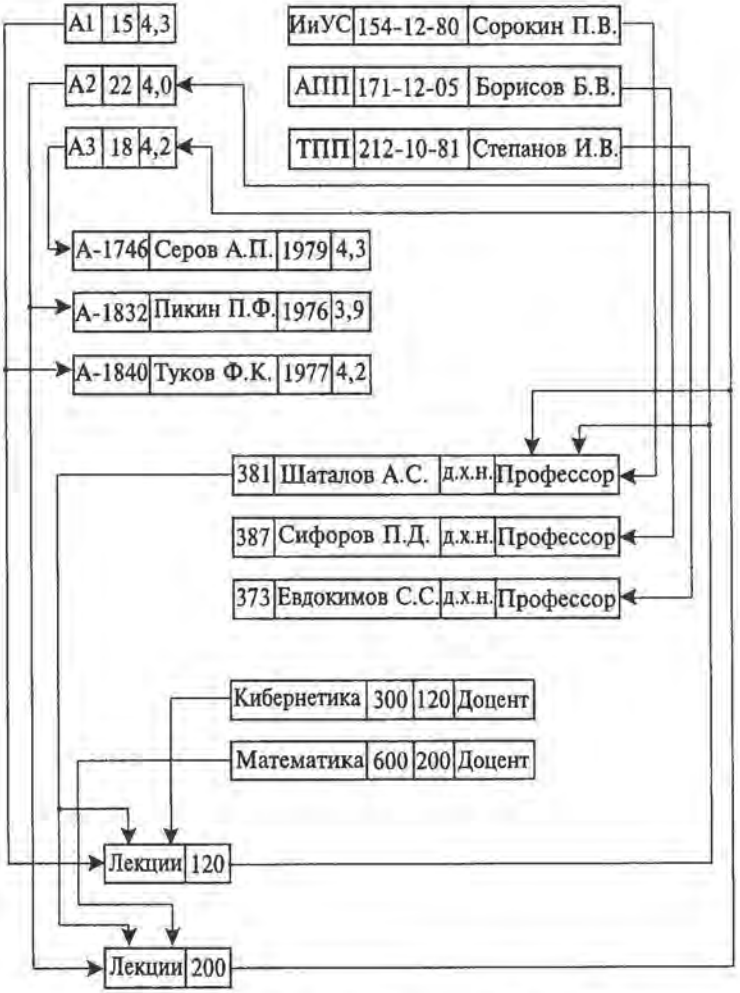


Рис. 6.5. Экземпляры логических записей

MANDATORY AUTOMATIC
ASCENDING KEY IS WS in WH

DUPLICATES NOT ALLOWED

SET OCCURRENCE SELECTION IS THRU
LOCATION MODE of OWNER

Базовыми языками в сетевых СУБД могут быть COBOL, PL/I. В частности, для СУБД DBMS базовым языком служит COBOL, а соответствие его команд командам языка БД приведено в табл. 6.1.

Таблица 6.1

Соответствие команд языков БД и COBOL

Язык БД	COBOL
AREA	REALM
OPEN	READY
CLOSE	FINISH
FIXED	HET
MANDATORY	PERMANENT
OPTIONAL	TRANSIENT
AUTOMATIC	AUTOMATIC
MANUAL	MANUAL
INSERT	CONNECT
REMOVE	DISCONNECT
MODIFY	MODIFY
STORE	STORE
DELETE	ERASE

В физической модели выделено понятие «схема» — совокупность типов записей и связей между ними.

База данных физически состоит из областей (AREA), которые разделены на блоки, называемые страницами. Размер страницы, каждая из которых имеет уникальный номер в одной и той же области, определяется пользователем. Номер страницы и строки в ней

образует физический ключ записи, а логическая структура реализуется с помощью указателей. Поиск в логической структуре может осуществляться по ключу, что в физической структуре реализуется указателями. Указатели (абсолютные или относительные) могут иметь цепочечную организацию (от записи к записи) или в виде массива указателей.

Создание сетевой БД (ЯОД)

Иллюстрацию ЯОД удобно провести с помощью программы, приведенной ранее.

Программа определяет имена схемы (SCHEMA), области (AREA) возможно и с объявлением начальной и конечной страниц. В область возможно копирование (CALL) из других схем. Задаются записи (RECORD) и вводится метод их размещения (LOCATION MODE IS ..), определяющий способ хранения. Различают методы DIRECT, CALC, VIA SET, SYSTEM, INDEX.

Метод DIRECT самый быстрый: управление осуществляет пользователь путем присваивания адресов. В последних версиях этот метод исключен.

Метод CALC (ключ указывается после слова USING) предусматривает кэширование. Подробно оно будет рассмотрено в физической БД, а здесь покажем лишь его суть.

При кэшировании адрес записи вычисляется по заданному ключу с помощью специальных алгоритмов (кэш-функций). Записи, приобретающие в результате кэширования один адрес (синонимы), размещаются на одной странице, что резко ускоряет процедуру доступа.

Размещение через набор VIA (SET) применяется, когда запись-член направляется на ту же страницу, где находится запись-владелец. Метод SYSTEM применяется по умолчанию, а метод INDEX используется редко.

Указывается возможность наличия (DUPLICATES) повторяющихся данных, расположенных в начале (FIRST) или конце (LAST) записей-членов, либо их запрет (NOT).

Необходимо указать и связи набора: запись-владелец (OWNER), запись-член (MEMBER), размещение записи-члена после экземпляра записи-владельца (FIRST), после последней записи-члена (LAST), после текущей записи (NEXT), перед текущей записью (PRIOR), отсортированные записи (SORTED), тип указателей (NET, PRIOR).

Предусматриваются своеобразные процедуры включения (и исключения) записей (рис. 6.6), более подробно описываемые в ЯМД.

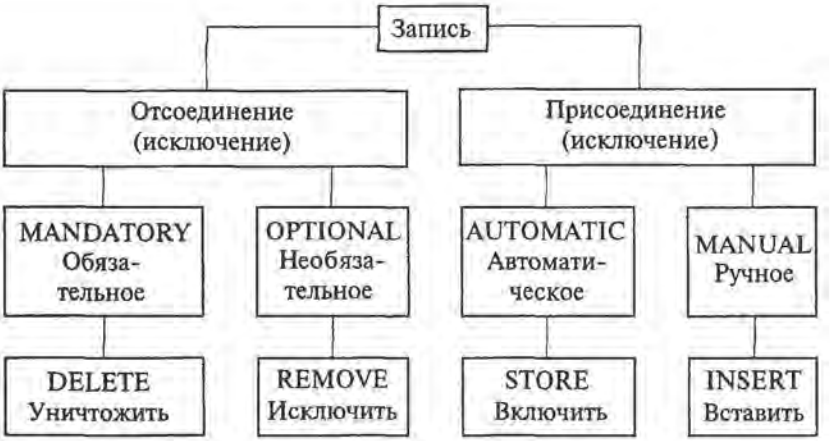


Рис. 6.6. Процедуры присоединения и отсоединения записей на языке СУБД DBMS

Устанавливается выбор экземпляра набора, в частности, по способу размещения владельца (SET OCCYRENCE SELECTION IS THRU LOCATION MODE of OWNER).

Может устанавливаться и блокировка монопольная (EXCLUSIVE) и немонопольная (KEEP).

Использование сетевой БД (ЯМД)

Команды ЯМД возможно разделить на несколько групп.

Открытие и закрытие БД производится командами OPEN и CLOSE.

Доступ осуществляется командами FIND, GET, OBTAIN. Команда FIND (FIND RECORD <имя записи> DB_KEY <ключ БД, точка входа>) осуществляет поиск экземпляров, GET — пересылку записи в промежуточную память. Команда OBTAIN «совмещает» функции двух предыдущих команд. У всех этих команд имеется несколько форматов.

Модификация данных проводится командой MODIFY.

Сложнее другие процедуры обновления, связанные с присоединением и отсоединением записей (рис. 6.6). Дело в том, что для изменения структуры связей их необходимо сначала разорвать, а затем установить вновь.

При отсоединении (MANDATORY) данные уничтожаются (DELETE), а при исключении (OPTIONAL) сохраняются (REMOVE:

REMOVE <имя записи> RECORD FROM <имя набора> SET) возможно и в виде сингулярного набора. Аналогично при ручном (MANUAL) присоединении, осуществляемом пользователем, например, при придании владельца сингулярному набору необходимо установить связь в наборе (INSERT: INSERT <имя записи> RECORD INTO <имя набора> SET). При автоматическом (AUTOMATIC) включении, выполняемом (FROM <имя набора> SET) СУБД, запись помещается в набор (STORE: STORE <имя записи> RECORD).

В заключение отметим, что в сетевой МД сложно строить БД и получать доступ к ней, непросто построить удобный интерфейс пользователя. К тому же МД не обладает должной адаптацией к произвольной структуре запроса.

6.3. ЛОГИЧЕСКАЯ СТРУКТУРА ИЕРАРХИЧЕСКОЙ БД

Иерархическая модель данных была исторически первой структурой БД, видимо, из-за того, что древовидные иерархические структуры широко используются в повседневной человеческой деятельности. Это всевозможные классификаторы, ускоряющие поиск требуемой информации, иерархические функциональные структуры управления. Наиболее известной иерархической СУБД до сих пор остается IMS [1—11].

Элементами иерархической модели являются поле и сегмент. Структурные связи определяются подчиненностью, в силу чего сегмент более низкого уровня называют порожденным, а более высокого уровня — исходным. Сегмент самого верхнего уровня носит название корневой. Именно через него осуществляется доступ к иерархической БД (точка входа). Экземпляр порожденного сегмента не может существовать в отсутствие экземпляра исходного сегмента. В силу этого положения при удалении экземпляра исходного сегмента удаляются и все экземпляры порожденных сегментов.

Если структура запроса совпадает со структурой иерархической БД, то такая модель данных обладает самым высоким быстродействием и потому чаще всего применяется в суперЭВМ. В противном случае быстродействие может резко снизиться или данные совсем могут быть не получены. Чтобы избежать последнего неприятного случая, в иерархической МД вводят еще один тип связи — логической, о которой поговорим позднее. Иерархические, как и сетевые, МД могут быть представлены в виде графов: сегмент отражается в виде узла.

В графической диаграмме схемы базы данных вершины графа также используются для интерпретации типов сущностей, а дуги —

типов связей между типами сущностей. При реализации каждая вершина графа представляется совокупностью описаний экземпляров сущности соответствующего типа.

В иерархической модели данных действуют более жесткие внутренние ограничения на представление связей между сущностями, чем в сетевой модели. Основные внутренние ограничения иерархической модели данных:

- все типы связей функциональные (1 : 1, 1 : M, M : 1);
- структура связей древовидная.

Результатом действия этих ограничений является ряд особенностей процесса структуризации данных в иерархической модели.

Древовидная структура (дерево) — это связный неориентированный граф, который не содержит циклов (петель) из замкнутых путей. Обычно при работе с деревом выделяют конкретную вершину, которую определяют как корень дерева, и рассматривают ее особо: в нее не заходит ни одно ребро. В этом случае дерево становится ориентированным. Корневое дерево можно определить следующим образом:

1) имеется единственная особая вершина, называемая корнем, в которую не заходит ни одно ребро;

2) во все остальные вершины заходит только одно ребро, а исходит произвольное количество ребер;

3) не имеется циклов.

В программировании используется другое определение дерева, позволяющее при решении задач рассматривать дерево как структуру, состоящую из меньших деревьев (или поддеревьев), т. е. как рекурсивную структуру.

Рекурсивно дерево определяется как конечное множество T, состоящее из одного или более узлов (вершин), таких, что:

1) существует один специально выделенный узел, называемый корнем дерева;

2) остальные узлы разбиты на $m >= 0$ непересекающихся подмножеств T_1, T_2, ..., T_m, каждое из которых в свою очередь является деревом.

Деревья T_1, T_2, ..., T_m называются поддеревьями корня.

Из определения следует, что любой узел дерева является корнем некоторого поддерева, содержащегося в полном дереве. Число поддеревьев узла называют степенью узла. Узел называется концевым, если он имеет нулевую степень. Иногда концевые узлы называют листьями, а ребра — ветвями. Узел, не являющийся ни корнем, ни концевым узлом, называется узлом ветвления.

Таким образом, иерархическая древовидная структура, ориентированная от корня, удовлетворяет следующим условиям:

• узел состоит из одного или нескольких атрибутов;

• иерархия всегда начинается с корневого узла;

• на верхнем уровне может находиться только один узел — корневой;

• на нижних уровнях находятся порожденные (зависимые) узлы: они могут добавляться в вертикальном и горизонтальном направлениях без ограничения;

• каждый порожденный узел, находящийся на уровне i, связан только с одним непосредственно исходным узлом (непосредственным родительским узлом), находящимся на более высоком уровне $(i-1)$ иерархии дерева;

• каждый исходный узел может иметь один или несколько непосредственно порожденных узлов, которые называются подобными (связи 1 : M);

• доступ к каждому порожденному узлу выполняется через его непосредственно исходный узел;

• существует единственный иерархический путь доступа к любому узлу, начиная от корня дерева (рис. 6.7), например путь ABEI.

Поскольку узлы, входящие в иерархический путь, могут встретиться не более одного раза, иерархические пути в древовидной структуре линейны. Любой узел, находящийся на иерархическом пути выше рассматриваемого узла, является для последнего исходным узлом (родительским). Любой узел, находящийся на иерархическом пути ниже рассматриваемого узла, является для него порожденным узлом.

В иерархических моделях данных используется ориентация древовидной структуры от корня, т. е. дуги, соответствующие функци-

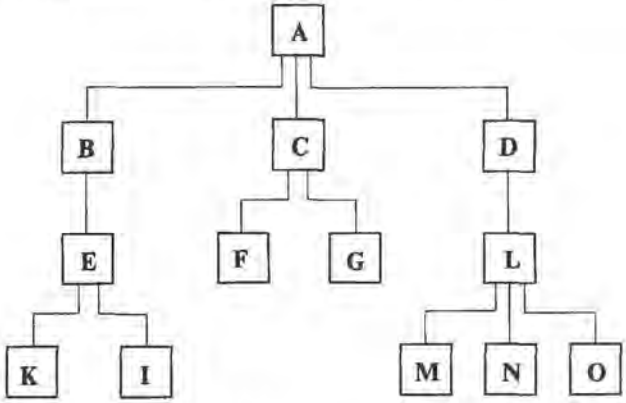


Рис. 6.7. Иерархический путь доступа

ональным связям, направлены от корня к листьям дерева. Графическая диаграмма схемы базы данных для иерархической базы данных называется деревом определения.

Вершины дерева определения БД соответствуют введенным типам групп записей, с помощью которых выполнена интерпретация типов сущностей. Обычно допускают только простые типы групп, т. е. группа, представляющая вершину дерева определения, не должна включать составные и повторяющиеся группы, но их можно представить как самостоятельные вершины дерева определения. Корневой вершине дерева определения соответствует тип корневой группы, остальным вершинам — типы зависимых групп. Дуга дерева определения, соответствующая групповому отношению, представляет некоторый тип связи между рассматриваемыми типами сущностей (которые представлены соответствующими типами групп). Дуга исходит из типа родительской (исходной) группы и заходит в тип порожденной группы. Дуги обычно называют связью «исходный—порожденный». Поскольку между двумя типами групп может быть не более одной такой связи, то на графической диаграмме схемы иерархической базы данных связи могут специально не помечаться. Тип зависимой группы можно идентифицировать соответствующей последовательностью связей «исходный—порожденный». Иерархический путь в дереве определения представляется последовательностью групп, начинающейся типом корневой группы и заканчивающейся типом заданной группы.

Следствием внутренних ограничений иерархической модели является то, что каждому экземпляру зависимой группы в иерархической БД соответствует уникальное множество экземпляров родительских групп (по одному экземпляру каждого типа родительской группы).

Иерархия — это разновидность сети, являющаяся совокупностью деревьев (лесом), в которой все связи направлены от отца к сыну. Рассматривая этот тип моделей, будем использовать терминологию сетевых моделей, введя дополнительное понятие — тип виртуальной логической записи (необходим, когда надо поместить какой-либо тип записи в два или более деревьев иерархии или в нескольких местах в одном дереве). Такая логическая запись представляет собой указатель на логическую запись некоторого типа и устраняет избыточность.

Снова обратимся к рис. 3.4, *в* (см.). Теперь придется переместить некоторые поля в другие таблицы. На рис. 6.8 показана прямая связь ряда таблиц.

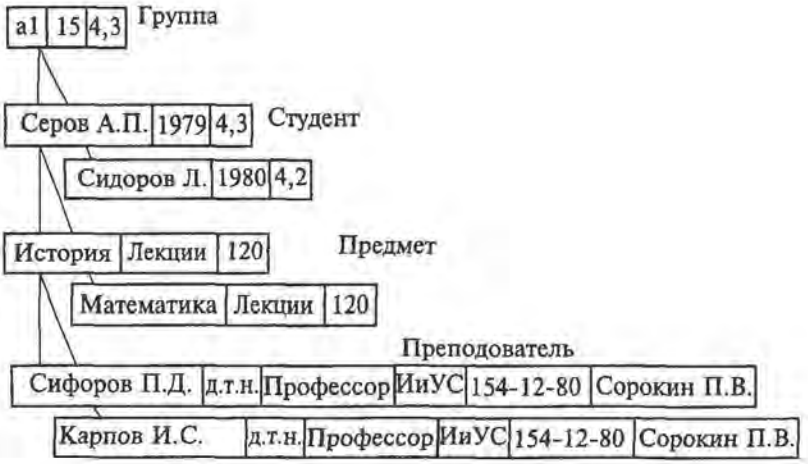


Рис. 6.8. Структура иерархической БД

6.4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИЕРАРХИЧЕСКОЙ БД

В иерархической МД имеется сильная зависимость логической структуры от физической, что хорошо видно из рис. 6.9 и программы на языке СУБД IMS, приведенной ниже.

```
      DBD NAME=DB3, ACCESS=HISAM
DATASET DO1=DEPTDO1, DEVICE=3380, OWFLW=DEPTOVF
SEGM NAME=PART, BYTES=32
LCHILD NAME=(COMP_ASSEMB, DB3), PAIR=ASSEMB_COMP
FIELD NAME=(PS, SEQ), BYTES=5, START=1
FIELD NAME=PD, BYTES=25, START=6
FIELD NAME=CL, BYTES=2, START=31
SEGM NAME= ASSEMB_COMP, BYTES=10
POINTER=(LPART, TWIN, LTWIN)
PARENT(PART), SOURCE(PART, PHISICAL, DB3)
FIELD NAME=(PS, SEQ), BYTES=5, START=1
FIELD NAME=OTY, BYTES=5, START=6
SEGM NAME= COMP_ASSEMB, BYTES=10
POINTER=RAIDER
FIELD NAME=(PS, SEQ), BYTES=5, START=1
PARENT PART, SOURCE(ASSEMB_COMP, DB3)
FIELD NAME=(PS, SEQ), BYTES=5, START=1
FIELD NAME=OTV, BYTES=5, START=6
```

Рис. 6.9. Реализация иерархической БД

На внутреннем уровне древовидные структуры могут быть представлены различными способами. Например, отдельный экземпляр структуры, соответствующий схеме базы данных, можно определить как экземпляр записи файла. В этом случае иерархическая база данных на внутреннем уровне будет представлена одним экземпляром этого файла. Многие иерархические СУБД могут поддерживать несколько различных баз данных. В этом случае каждая БД на внутреннем уровне представляется одним файлом, который объединяет экземпляры записей одного типа со структурой, соответствующей схеме этой базы данных. На рис. 6.9 приведен пример схемы иерархической базы данных и ее возможной реализации на внутреннем уровне.

Создание иерархической БД (ЯОД)

Базовыми языками иерархической БД по-прежнему может быть COBOL, PL/I, Assembler.

Иллюстрацию ЯОД удобно вести на основе программы на языке СУБД IMS. Объявляются имена БД, сегмента и полей. Для одного из полей, по которому идет упорядочение, указывается метка SEQ и количество порожденных сегментов: один (U) или много (M). Отмечается исходный (PARENT) сегмент. Может присутствовать поле указателя (POINTER), метод доступа (HISAM).

Введение логических связей обозначается командой LCHILD, в которой указываются имена порожденного логического сегмента и сегмента БД, а в команде PARENT — исходные физический и логический сегменты БД.

Использование иерархической БД (ЯМД)

Обновление осуществляется командами REPLACE (заменить) и DELETE (удалить текущий сегмент и порожденные). Вставка (INSERT) может быть по отношению к текущему исходному сегменту (INSERT S) или по указанию (INSERT клиенты WHERE ГОРОД = 'СПб' and ИМЯ АГЕНТА= 'Святослав').

Помещение сегмента в область ввода—вывода осуществляется командой GET:

```
GET UNIQUE <имя сегмента>
WHERE <условие>,
GET NEXT,
GET NEXT WITHIN PARENT.
```

Из описания иерархической МД видно, что БД построить достаточно сложно. Здесь нет четкого разделения логической и физической структур. Набор структур запросов ограничен, а для неиерархических связей требуются дополнительные логические связи, превращающие фактически иерархическую МД в сетевую.

Контрольные вопросы

1. Каковы структурные элементы сетевой модели данных?
2. Что такое элемент данных? агрегат? запись?
3. Назовите виды агрегатов.
4. Как обеспечивается связь между записями?
5. Каковы разновидности наборов?
6. Каковы правила построения сетевой БД?
7. Почему нельзя реализовать отношение M : N? Как оно реализуется?
8. Каковы структурные элементы иерархической модели данных?
9. Каковы типы сегментов?
10. Как обеспечивается двусторонняя связь между сегментами?
11. Как обеспечивается доступ к сетевой БД?

Глава 7

Объектно-ориентированные базы данных

Серьезные недостатки реляционной модели данных привели к необходимости поиска других моделей. Такой прогрессивной и перспективной моделью данных является объектно-ориентированная модель данных.

В ней собственно база данных, интерфейс пользователя и алгоритм приложения построены с использованием объектно-ориентированного подхода.

Суть объектно-ориентированных баз данных иллюстрируется на СУБД CACHE, получившей распространение в России. В этой СУБД, чтобы получать данные из многочисленных реляционных БД, предусмотрен объектный доступ (объектно-ориентированная модель) и SQL-доступ (реляционная модель с использованием языка SQL2). Хранение данных в CACHE осуществляется с помощью многомерной модели данных, позволяющей уменьшить объем потребной памяти при одновременном увеличении скорости доступа к данным.

7.1. НЕДОСТАТКИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Автор реляционной модели данных Э.Ф. Кодд первоначально сформулировал 12 требований к БД, чтобы она могла называться реляционной. В дальнейшем этот перечень увеличился до 333 требований. Им, несмотря на широкое распространение реляционных баз данных, не удовлетворяет ни одна из известных СУБД.

Более того, при значительных объемах данных начинают проявляться недостатки реляционных баз данных. К этим недостаткам относятся: сложность структуры, вызванная необходимостью проведения нормализации; низкая производительность из-за поиска по ключу, что в 3—5 раз увеличивает количество операций доступа; ограниченный набор типов данных; представление данных только в виде двумерных таблиц и невозможность реализации таблиц с нелинейной структурой; невозможность послойного рассмотрения данных (например, «Работающие» — в одном слое, «научные сотрудники» и «преподаватели» — в другом, подчиненном слое); нестыковка

с принципами все более широко применяемого объектно-ориентированного подхода; невозможность задать для определенного типа данных набор операторов-методов, которые приходится вводить в конкретном приложении; возникновение *конфузии* — утраты при многочисленных обновлениях третьей (а порой и второй) нормальной формы; сложность совмещения с другой парадигмой хранилищ данных.

Одним из способов устранения указанных недостатков является построение объектно-ориентированной базы данных (ООБД). Ее появление стимулировано и требованиями большой быстродействующей памяти (свыше 20 Гбайт) для систем конструирования/производства (CAD/CAM).

7.2. СОСТОЯНИЕ РАЗВИТИЯ ООБД

В соответствии с «Манифестом ООБД» [2], опубликованном в 1989 г., используется формула

ООСУБД = СУБД + ООЯП,

где сокращения ОО и ЯП означают объектно-ориентированный язык программирования. ООСУБД должна поддерживать объекты с нелинейной структурой (сложные объекты, в том числе с иерархией типов), что достигается инкапсуляцией и наследованием. Легко поддерживается расширяемость, вычислительная полнота, языки запроса.

В 1991 г. была сформирована группа Object Database Management Group (ODMG), перед которой была поставлена цель построить стандарты для ООБД хотя бы на уровне стандартов для реляционных БД. В 1993 г. эта группа предложила своеобразный стандарт для ООБД, названный ODMG-3, который включал:

1) объектную модель Object Data Model (*ODM*);
2) язык определения объектов Object Definition Language (ODL);
3) объектный язык запроса Object Query Language (OQL);
4) интерфейсы языков программирования (C^{++} и других).

В настоящее время насчитывается свыше 300 объектно-ориентированных СУБД (ООСУБД), данные некоторых приведены в табл. 7.1. Сфера их применения указана в табл. 7.2.

Из табл. 7.1 видно, что ООСУБД создаются с использованием различных подходов.

Выбор ООСУБД может определяться наличием поддержки реляционных БД; интерфейса с языками С и расширениями SQL;

Таблица 7.1

Характеристики некоторых ООСУБД

Фирма-производитель	Название ООСУБД	Средства разработки	Подход к разработке
Objectivity	Objectivity/DB	C, C++, SQL, Java	Расширение объектно-ориентированных библиотек классов
Poet Software	Poet	C, C++, ODBC, Java	
Object Design	Object Store	C, C++, Java	
Ontos Inc.		C++, Java	
Versant Object Technology	Ontos DB, Versant	C++, Java	
Computer Associate	Jasmine	C++, Java	
НПЦ «Интелтек Плюс»	ODB-Jupiter	C++	
O2 Technology	O2	C++, Java	Вставка объектно-ориентированного языка БД в обычный базовый язык
GemStone Inc.	GemStone	C++, Java	Расширение языка (C++) возможностями работы с БД
InterSystems	CACHE	Semantic Information Manager, Cache ObjectScript	Новый язык базы данных или модели данных

средств разработки и администрирования; доступа к данным из действующих БД (с помощью ODBC и SQL-запросов); возможности работы с различными платформами.

Фирма BKS Software предлагает ООСУБД Poet как пользовательскую, переносимую, интегрированную с двумя ступенчатыми механизмами транзакций в средах Windows NT, Unix с использованием базового языка C++.

В России — при активной работе фирмы СП.АРМ (С.-Петербург) — все шире используется СУБД Cache фирмы InterSystems, апробированная и хорошо проявившая себя за рубежом, прежде всего для таких ответственных учреждений, как банки.

Таблица 7.2

Сфера применения ООСУБД

	Versant	GemStone	O2	ObjectStory	POET
Моделирование	+	+	+	+	+
САПР	+	+	+	+	+
Управление производством	—	+	+	+	+
Обработка изображения	+	+	—	+	+
CASE	+	+	—	+	+
Планирование	+	—	—	—	—
Гипертекстовые системы	+	—	+	+	+
Издательство	—	—	—	+	—
Экспертные системы	+	—	—	+	Нет информации

Следует заметить, что ООСУБД все чаще применяют как составную часть другого приложения.

Например, компания Computervision, производящая программное CAD-обеспечение, интегрировала в свой продукт СУБД ObjectStory.

Компания Enterprise Integration Technology предлагает продукт MKS, позволяющий вести разработку технологических процессов и оборудования; управление предприятием; проектирование производственных помещений; диагностику, мониторинг (отслеживание); моделирование и планирование.

Американские фирмы Aototrol Technology, Step Tools, Dec используют ООСУБД ObjectStory для работы со слабо структурированными данными в стандарте Standard of Exchange of Product model data (STEP) обмена данными.

7.3. СУЩНОСТЬ ООБД

Суть объектно-ориентированной БД определяется объектно-ориентированным подходом (рис. 7.1), который подразумевает объектно-ориентированное проектирование и объектно-ориентированное программирование.

Рис. 7.1. Объектно-ориентированный подход

Объектно-ориентированное проектирование — методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логических и физических, а также статических и динамических моделей проектируемой системы.

Объектно-ориентированное программирование — методология программирования, основанная на представлении программ в виде связанной совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию по наследованию.

Объектно-ориентированное проектирование предполагает не только деление (декомпозицию) базы знаний или базы данных на составные части (рис. 7.2), но и рассмотрение общей этапности реализации БД, выбор инструмента реализации с учетом оговоренных в техническом задании вариантов реализации.

Рис. 7.2. Объектно-ориентированное проектирование

Объектно-ориентированное программирование берет за основу модель атомарного элемента (рис. 7.1).

Дело в том, что, несмотря на мощные средства отладки программ, для повышения производительности процедуры программирования целесообразно отлаживать программы последовательно, по блокам. Программа в целом отлаживается быстрее, если блоки были отлажены заранее. Эти предпосылки и положены в основу объектно-ориентированного программирования.

Любая математическая модель имеет вид, представленный на рис. 7.1, а сложная — на рис. 7.3.

Выделяется несколько специфических понятий. *Класс* — это фактически некоторый блок со структурой (рис. 7.1). Однако здесь используется несколько другая терминология. Данные называют *свойствами*, а алгоритмы — *методами*. Доступ к классу осуществляется через свойства — в статическом режиме написания и отладки программы или через методы — в процессе выполнения (работы) программы.

Начало работы класса задается с помощью специальных внутренних (например, нажатие кнопки) или внешних (вызов из другой подпрограммы) сигналов, называемых *событиями*.

Программную реализацию класса называют компонентой. Реализация компоненты в некоторой прикладной программе получила название объекта.

В объектно-ориентированном программировании используются три основных принципа: инкапсуляция, наследование, полиморфизм.

Инкапсуляция — выделение класса с доступом к нему через свойства или методы.

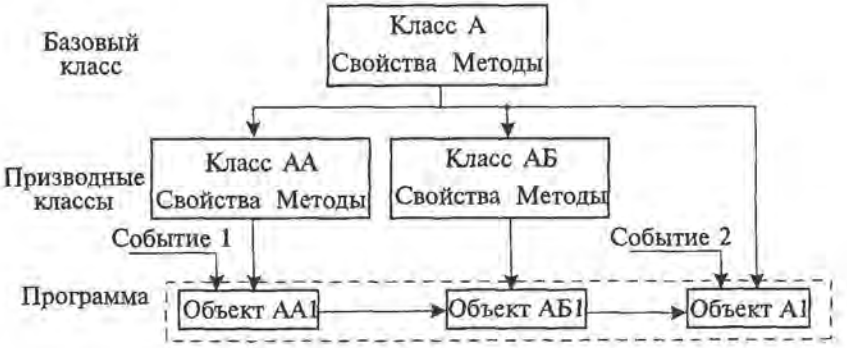


Рис. 7.3. Объектно-ориентированное программирование

Наследование — трансформация класса путем изменения свойств и методов с помощью методов, называемых конструктором и деструктором.

Все классы (компоненты) строятся по иерархическому принципу с происхождением от некоторого исходного класса.

Полиморфизм позволяет использовать метод с одним именем как в базовом, так и в производных классах. Дело в том, что количество классов значительно: уже сейчас оно приближается к ста и продолжает расти. Если для производных классов применять для однотипных, «похожих» методов (например, начертание квадрата и окружности) разные имена, пользователю, особенно начинающему, будет сложно освоиться с таким разнообразием имен.

Приведем основные положения ООБД, базирующиеся на объектно-ориентированном подходе.

1. В качестве значения столбца может быть использован произвольный кортеж. Иными словами, столбец может являться как бы некоторой другой таблицей. Таким образом, создается возможность реализации таблиц с нелинейной структурой.

2. Процедура манипуляции данными позволяет присоединять процедуры, определенные значениями столбцов.

3. Данные столбцов могут наследоваться.

4. Элементами отношений могут служить не только отдельные элементы, как в реляционных БД, но и множества.

5. Формируются классы данных, которые организуют столбцы в иерархию.

Базовым языком ООБД чаще всего является C++. Для работы с такими ООБД разрабатывается новый вариант языка программирования SQL, получивший название SQL3 и содержащий внутри себя в качестве частного случая реляционную модель (SQL2). Если SQL2 определяет семь способов связывания со стандартными языками программирования, в SQL3 это количество предполагается увеличить.

В технологии разработки ООБД конкурирует два направления.

- Distributed Object Linking and Embedding (OLE) фирмы Microsoft.
- Common Object Request Broker Architecture (CORBA) группы OBDMG, поддерживаемая фирмами IBM, Novell, DEC, с ориентацией на все платформы. В рамках этого направления выделены и сформированы указанные ранее язык определения объектов Object Definition Language (ODL); объектный язык запроса Object Query Language (OQL); язык определения интерфейсов Interface Definition Language (IDL).

Во втором направлении обычно выделяют [4] два стандарта управления БД:

• ODL/OQL, в котором объекты и методы формируются обычно с помощью языка программирования C;

• язык программирования SQL3, некоторые особенности которого описаны в гл. 9.

Необходимо отметить, что к объектно-ориентированным возможно отнести только те базы данных, у которых все структурные элементы реализации (см. рис. 7.2) построены с использованием объектно-ориентированного подхода. Если хотя бы один структурный элемент реализации не использует объектно-ориентированный подход, говорят об *объектно-реляционной базе данных* (ОРБД), которые будут представлены подробнее в гл. 8.

Таким образом, чтобы воспользоваться объектно-ориентированным подходом в построении собственно БД, необходимо:

1) провести инкапсуляцию данных, т. е. выделить классы и объекты;

2) определить возможные виды структуры реализуемых таблиц;

3) создать наследование классов данных;

4) обеспечить полиморфизм.

Реализация даже первой позиции неоднозначна и различна, например в ООСУБД и ОРСУБД. Имеется некоторое отличие даже в рамках различных ООСУБД.

В различных ООБД существует несколько отличный, по крайней мере по терминологии, набор свойств. В то же время существенного различия между наборами свойств нет, и потому иллюстрируем их на примере ООСУБД CACHE.

В ней выделяют классы типов данных и классы самих данных (классы объектов, таблицы).

Если провести аналогию с терминами реляционной БД, то понятию «класс (объект)» ООБД соответствует примерно понятие «таблица», а понятию «свойство» — «поле (столбец) таблицы».

В качестве свойств [38, 40] приняты константы, ссылки на объекты, встроенные объекты, потоки данных (BLOB и CLOB), коллекции в виде массивов (Array) и списков (List), многомерные свойства. К свойствам относят и связи между объектами.

Для свойств имеются типы, основными из которых являются Currency (денежный), Date, Float, Integer, List (список), Name (имя), Numeric, Status (ошибки), String.

В качестве типов свойств могут выступать связи, которые [40] можно разделить на простые, реализуемые ссылками, и отношения, иногда называемые ассоциациями. При ссылке в качестве типа свойств выступает класс (объект), с которым связывается данное свойство. Возможно встраивание одного объекта в другой. В ассо-

циациях выделяют отношения ONE-TO-MANY (1:M) и PARENT—CHILDREN (родитель — дети).

Различают три основные группы методов: код (на языке Cache ObjectScript), выражение (фрагмент кода, вызываемый по значению), вызов. Нетрудно видеть, что данные ячеек *неатомарны*.

Структура ООБД может быть построена различными способами: с помощью визуального языка, языков программирования Cache ObjectScript, SQL.

В других ООСУБД для этих целей часто используют язык C^{++}.

ООБД в рамках системы OLTP и OLAP взаимодействует с многомерной моделью данных (ММД). Обсудим специфику ММД.

7.4. МНОГОМЕРНАЯ МОДЕЛЬ ДАННЫХ

Вообще говоря, ММД может взаимодействовать и с реляционной БД, потому данный параграф уместен и в гл. 5. Однако взаимодействие с объектно-ориентированной моделью предпочтительнее, и потому рассмотрим ММД здесь.

Многомерная модель используется, как отмечалось ранее, в хранилищах данных. Имеются три разновидности ММД: MOLAP, ROLAP, HOLAP.

MOLAP предполагает формирование так называемого многомерного куба (гиперкуба). Трехмерный куб показан на рис. 7.4.

Каждая из координат куба называется *измерением*. Измерения реализуются с помощью индексов, которые, как известно, позволяют резко увеличить скорость доступа к данным. По разным оценкам, эта скорость в 10—100 раз выше, чем в реляционных моделях данных. Значение результата в многомерной модели помещается на пересечении (в ячейке) соответствующих значений измерений.

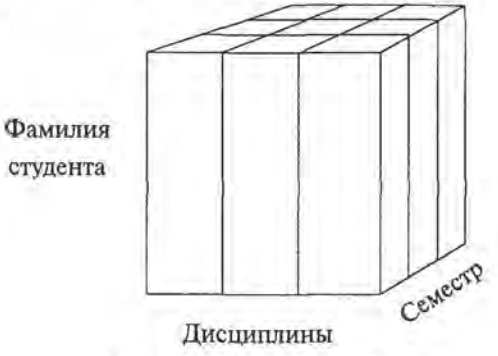


Рис. 7.4. Трехмерный куб

Аналогом такой модели можно считать задание функции, например, $z = f(x, y)$. В ММД координаты x и y суть измерения, а z — значение-результат. Как и в функции нескольких переменных, в ММД должна быть предусмотрена возможность «движения» как внутри одного измерения, так и перехода с одного измерения на другое (например, от «линии» к «плоскости»).

За увеличенную скорость доступа приходится платить увеличенным объемом памяти. Соотношение между полезным и потребным объемами памяти в многомерной модели данных в 5—10 раз больше, чем в реляционной модели данных. Кроме того, в ММД много пустых ячеек, и требуются специальные эффективные программы хранения значения NULL. Желательно удаление пустых ячеек. В многомерной модели выполняются четыре основные операции: агрегация (свертка) данных; дезагрегация (детализация) данных; сечение (при фиксированных значениях одного или нескольких измерений); вращение (для двумерного случая это аналог транспонирования матрицы).

В **ROLAP** используются реляционные «составляющие». Возможны две схемы структуры: «звезда» (рис. 7.5) и «снежинка» (рис. 7.6).

Рис. 7.5. Схема «Звезда»

Рис. 7.6. Схема «Снежинка»

В любой схеме выделяются одна фактологическая таблица, в которой, собственно, и хранятся данные, и несколько справочных таблиц, каждая из которых характеризует одну из размерностей куба. При использовании схемы «звезда» требуется проведение денормализации данных.

ROLAP позволяет дать характеристику размерности хранилища данных (табл. 7.3).

Таблица 7.3

Размерность хранилища данных

Размерность	Предельный объем, Гбайт	Число строк в фактологической таблице, млн
Маленькая	3	1—10
Средняя	25	11—100
Большая	200	101—999
Сверхбольшая	Более 200	1000 и более

Сравнение MOLAP и ROLAP дает следующие результаты:

• MOLAP обладает высоким быстродействием, но возникают проблемы с хранением больших объемов данных;

• ROLAP не имеет ограничений на объем данных, однако обладает гораздо меньшим быстродействием.

Отсюда возникает идея построения HOLAP, совмещающего достоинства MOLAP и ROLAP.

HOLAP. Дело в том, что все данные ХД одновременно никогда не требуются. Каждый раз используется лишь их часть. В связи с этим целесообразно данные разделить на предметные подобласти, которые называют киосками (магазинами, витринами) данных. *Киоск данных* [17] — специализированное тематическое хранилище, обслуживающее одно из направлений деятельности фирмы.

В этом случае центральное хранилище может быть реализовано с использованием реляционной БД, а основные данные хранятся в многочисленных киосках.

7.5. CACHE КАК СИСТЕМА УПРАВЛЕНИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ БАЗОЙ ДАННЫХ

Покажем некоторые особенности построения объектно-ориентированной базы данных на примере СУБД CACHE.

Прежде всего обратим внимание на объектно-ориентированное (объектное) построение собственно БД в рамках СУБД CACHE, поскольку объектное построение интерфейса пользователя и алгоритма приложения подробно обсуждалось в гл. 1.

В СУБД CACHE используется «объектная» специфическая терминология, несколько отличная от системы определений объектно-ориентированного подхода.

Чтобы было ясно, о чем идет речь, сведем эту терминологию в таблицу (табл. 7.4), из которой видны особенности структуры ООБД (ООСУБД).

1. Ячейки таблиц, как и в расширенных ОРБД, неатомарны. В качестве ячеек могут выступать коллекции (списки, массивы), что соответствует спискам (List) и многомерным наборам (MULTISET) в расширенной объектно-реляционной БД.

2. Связи между классами устанавливаются через указатели (OID, OREF).

3. Общие подтаблицы нескольких таблиц формируются как встраиваемые объекты (аналог абстрактных классов ROW, и прежде всего — ROW TYPE объектно-реляционных БД).

4. Предусмотрена возможность хранения больших объектов.

Таблица 7.4

Сравнительная терминология

CACHE	Реляционные БД	Объектно-ориентированный подход
Класс	Таблица	Класс
Экземпляр класса (объект)	Строка	—
Идентификатор OID	Ключ	—
Свойство	Столбец, поле	Свойство
Ссылка на хранимый объект	Внешний ключ	—
Встраиваемый объект	Повторяющиеся поля, используемые в нескольких таблицах	—
Метод класса	Хранимая процедура	Метод
Коллекция-список	Столбец с ячейкой-списком	—
Коллекция-массив	Подтаблица	—
Запрос	Хранимая процедура или вид	—
Поток данных	BLOB	—

Рассмотрим построение собственно базы данных.

В соответствии с объектно-ориентированным подходом следует рассмотреть понятие «класс» и ассоциированные с ними термины (свойство, метод).

Понятие «класс» в СУБД CACHE достаточно многогранно (рис. 7.7). Прежде всего выделяют классы типов данных и классы самих данных (классы объектов, таблицы).

Термин «класс» в СУБД CACHE имеет два понимания: объект; множество (подмножество).

Классы типов данных (в том числе созданных пользователями) приписываются атрибутам (полям, свойствам) объектов (таблиц), получающим данные того или иного типа в качестве значений. Эти

Рис. 7.7. Виды классов

классы не могут содержать свойств и образовывать экземпляры, не имеют собственной идентификации (для ссылок), однако обладают методами. Каждый тип данных представляет собой класс. Классы типов данных могут создаваться и пользователями.

Имеются три формата типов данных: хранения в БД (storage), логический (logical) в памяти компьютера, отображения (display) на экране монитора. Форматы могут быть преобразованы друг в друга.

Методы классов типов данных предоставляются через интерфейс *типов данных (с их параметрами). Перечислим основные типы данных* (в логическом формате): %Library.String, %Library.Binary, %Library.Boolean, %%Library.List (формат списка), Library.Numeric, %Library.Integer, %Library.Float, %Library.Name (имя в формате «Фамилия Имя»), %Library.Date, %Library.Time, %Library.Currency (денежный).

Объектом называют экземпляр класса (строку) в отличие от реализации компонента в соответствующем контейнере (как, например, в программном продукте Delphi).

Класс объектов определяет структуру данных и поведение объектов одного типа. Класс объектов характеризуется именем класса, свойствами, методами.

В классах объектов по характеру процессов выделяют незарегистрированные и зарегистрированные классы.

Незарегистрированные классы не поддерживают полиморфизм и не обладают автоматическим управлением. Назначение идентификаторов (OID) и объектных ссылок (OREF) осуществляет пользователь.

Зарегистрированные классы поддерживают полиморфизм и управляются автоматически от системного класса %Library.RegisteredObject (аналог классу TObject в Delphi). Экземпляры этого класса существуют в памяти процесса временно (временные объекты).

С точки зрения хранения данных зарегистрированные и незарегистрированные классы делятся на хранимые и встраиваемые.

Хранимые классы объектов наследуются от системного класса %Library.Persistent и способны длительно храниться в памяти объекта. Экземпляры таких классов обладают *однозначными* объектными идентификаторами OID. Хранимый объект может использоваться и как свойство класса (столбец таблицы) другого объекта. Иными словами, возможна ссылка на этот объект, что соответствует связи 1 : M двух таблиц реляционной БД.

Встраиваемые классы объектов наследуют свое поведение от системного класса %Library.SerialObject и могут быть сохранены только в составе соответствующих хранимых объектов. Встроенный объект в памяти характеризуется объектной ссылкой, в базе данных хранится в последовательной форме (разновидность коллекции-массива) как часть хранимого объекта, при этом идентификатор OID отсутствует.

Построение классов объектов возможно в режиме диалога или с помощью языков программирования (командная строка, программа). В диалоговом режиме возможно использовать визуальный язык (аналог языка QBE для запросов) или сопровождающий его язык определения классов (CACHE Definition Language — CDL) — аналог языка SQL в реляционных БД.

В качестве *свойств* (рис. 7.8) выступают константы (независимо от класса типа), встроенные объекты, ссылки на объекты, потоки данных BLOB, коллекции, многомерные переменные, двунаправленные связи между хранимыми объектами.

Потоки данных BLOB имеют две разновидности для символьных (CHARACTERSTREAM) и двоичных (BINARYSTREAM) данных (аналогично CLOB и BLOB для гибридных ОРБД).

Коллекции могут быть списком (List Collection) и массивом (Array Collection). В их состав могут входить константы, встроенные объекты и ссылки, которые задаются соответственно %Library.ListOfDataTypes, %Library.ListOfObjects, %Library.ArrayOfDataTypes, %Library.ArrayOf Objects.

Рис. 7.8. Типы свойств

В коллекции-массиве упорядочение ведется по полю, принятому в качестве ключа.

В коллекции-списке в качестве ключа выступает позиция элемента в списке.

Связь является двунаправленной — взаимные ссылки между таблицами. Они гарантируют ссылочную целостность.

Имеются *временные* (буферная память) и *вычисляемые* свойства.

Методы, как уже отмечалось, связаны с типом данных. По умолчанию принимается тип данных String.

Аргументы метода по умолчанию передаются по значению, а для передачи по ссылке аргументу должен предшествовать символ &. Методы, как и свойства, могут быть определены как public или private.

Выделяют методы класса и методы экземпляров класса (ссылка — ##this). Последние используются чаще.

В объектной модели выделяют:

- метод-код (код программ на языке CACHE ObjectScript);
- метод-выражение (одно выражение кода программы), в котором передача параметров проводится по ссылке, использование макросов, «встроенных» фрагментов SQL не допускается;
- метод-вызов (подпрограмм);
- генератор метода, используемый при компиляции программы и классов (чтобы классами можно было пользоваться).

Свойства характеризуются именем класса, именем свойства, типом данных, ключевыми словами, параметрами для типов данных. Свойства могут быть открытыми (public) и закрытыми (private).

Свойство связано с набором методов, который имеет два источника:

1) класс свойств (методы доступа Get() и Set());
2) тип данных.

Для классов CACHE, наряду с такими понятиями, как «свойство», «метод», характерны специфические понятия «параметры класса», «запросы», «индексы».

Параметры класса используются при компиляции.

Запрос — операции с множествами экземпляров классов. Результат запроса доступен через специальный интерфейс обработки результатов запросов ResultSet. Запросы могут иметь форму хранимых процедур.

Индекс — путь доступа к экземпляру класса. Он используется для повышения скорости выполнения запросов. Индекс создается на основе одного или нескольких свойств (полей).

Взаимодействие описанных составных частей СУБД (БД) обеспечивается языками программирования.

В CACHE существует три вида доступа (рис. 7.9).

Объектный доступ осуществляется в диалоге или с помощью языка программирования CACHE ObjectScript.

В **прямом доступе** используется многомерная модель данных (ММД). Именно в многомерной структуре ядра БД хранятся данные.

Суть ММД фактически заключается в задании значений некоторой функции от множества координат.

Пусть координаты имеют текущие значения i, j, k, l соответственно, а значение «функции» в этой «точке» — A_{ijkl}. Тогда это значение можно определить многомерной матрицей или набором

$$<i, j, k, l, A_{ijkl}>.$$

В рамках СУБД CACHE этот факт записывается в виде

$$A(i, j, k, l) = A_{ijkl}.$$

Возникает вопрос реализации такой модели. Для этого используют B*-дерево — иерархическое B-дерево, в каждом узле которого

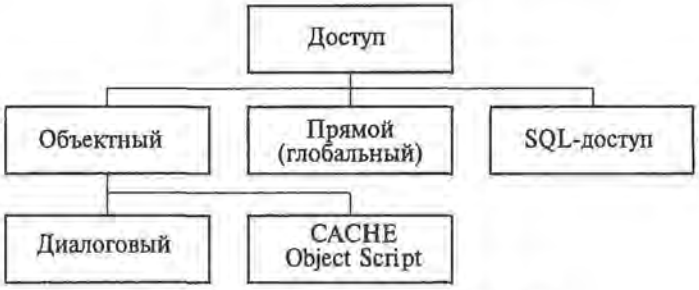


Рис. 7.9. Виды доступа в СУБД CACHE

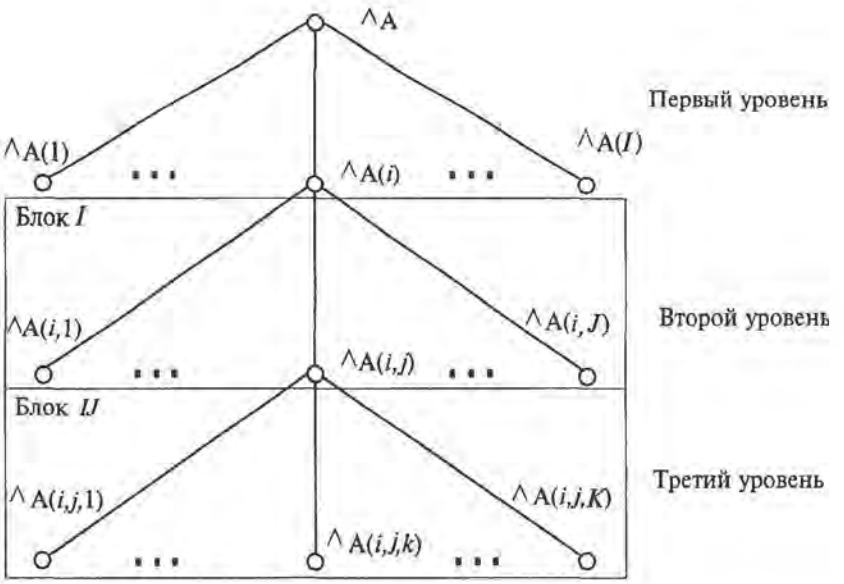


Рис. 7.10. B*-дерево

имеется блок. В этом случае данные в узлах представляются «координатами» и значениями (рис. 7.10).

В качестве «координат» удобно использовать индексы, а в рамках СУБД CACHE — глобальные переменные (*глобалы*).

ММД не рассматривается как инструмент реализации приложений, а служит основой хранения данных и используется при построении хранилищ данных (систем OLAP).

Особый интерес с позиций «перекачки» данных из реляционной СУБД в объектно-ориентированную представляет SQL-доступ. Основу **SQL-доступа** составляет язык программирования SQL2 (или SQL92). Этот язык предназначен для реляционных баз данных. СУБД CACHE, как показано в гл. 2, отличается по структуре от реляционных СУБД. Основные отличия связаны с неатомарностью ячеек БД. В таких ячейках может присутствовать встроенный класс (таблица) или коллекция.

К тому же необходимо реализовать процедуры наследования.

Таким образом, следует трансформировать язык SQL2, чтобы стало возможно оперировать с неатомарными ячейками и обеспечить процедуру наследования.

Именно по этому пути (а не по пути создания нового языка SQL3) пошли создатели СУБД CACHE.

Прежде всего следует отметить, что в рамках единой архитектуры CACHE каждый класс объектного представления является таблицей с тем же именем, а каждая таблица реляционной парадигмы — объектом. Соотношение объектов и реляционных аналогов приведено в табл. 7.4.

Заметим, что в реляционной БД нет точных аналогов методов и параметров классов и экземпляров классов, временных свойств. С другой стороны, в реляционном представлении имеют место триггеры, которые не нужны в объектной среде.

Экземплярам объектов соответствуют записи, идентифицируемые с помощью первичного ключа. Для хранимых объектов в качестве такого ключа возможно использовать идентификатор ID объекта (примерный аналог поля с типом данных «счетчик»).

Наследование в объектном представлении трансформируется в набор исходной и наследуемой таблиц.

По умолчанию имена объектного свойства и реляционного поля одинаковы. Переименование возможно с помощью ключевого слова SQLFIELDNAME в определении объектного свойства.

Объектные свойства-константы (атомарные ячейки) могут быть временными, многомерными и вычисляемыми. Вычисляемые свойства автоматически в таблицах не отображаются. В реляционной трактовке для этого следует предусмотреть вычисляемые поля. Последние могут быть трансформированы в вычисляемые свойства.

Удобнее вычисления размещать в методах класса, которые могут вызываться и из методов вычисления свойств, и как вычисляемые поля.

Ссылки на хранимые объекты осуществляются внешним ключом.

Встроенные объекты характеризуются составным именем поля с использованием символа подчеркивания (<имя класса, в который встроен объект>_<имя свойства>), т. е. Addres_Ulica.

Коллекция-список отображается в виде отдельного поля, содержащего список значений.

Коллекция-массив представляется в виде отдельной подтаблицы, связанной с основной таблицей через внешний ключ. Имя подтаблицы то же составное: <имя класса>_<имя коллекции>.

Объектным запросам соответствуют хранимые процедуры и виды (View). Ключевые слова — SQLPROC, SQLVIEW, SQLVIEWNAME.

Методы класса могут быть отображены в коде в виде вычисляемых полей или в виде хранимых процедур. Могут использоваться и триггеры, определяемые в объектном представлении на языке CDL или в рамках CACHE Object Architect.

Выделяют вложенный и встроенный языки SQL. Структура языка SQL2 (интерфейсного и вложенного) отображена во многих источниках.

В связи с этим акцентируем внимание на отличиях языка SQL в СУБД CACHE (CACHE SQL) от языка SQL2. Расширение **вложенного языка** SQL2 имеет место в следующих направлениях:

1) дополнительные операторы;
2) объект CURSOR;
3) поля-списки;
4) соединения (ссылки, отношения зависимости).

1. ***Дополнительными операторами являются:***

=* — внешнее соединение;

-> — неявное соединение;

_, # — конкатенация (имен) и целочисленный остаток от деления (модуль);

? — оператор проверки по шаблону;

[— оператор вхождения;

& — оператор И;

! — оператор ИЛИ;

] — оператор следования за.

Возможно использование как одинарных, так и двойных кавычек; приставки not перед логическими операторами (not =, not <, not >).

2. В дополнение к таким объектам, как таблицы, виды, хранимые процедуры, индексы, ограничения, генераторы, триггеры, в рамках CACHE используют объект SQL2, получивший название CURSOR.

CURSOR — некоторый набор данных, формируемый чаще всего оператором select. Он содержит несколько записей и отличается от хранимых процедур, которые только вычисляют набор данных, и вида, вычисляемого каждый раз при запросе к нему. CURSOR вычисляется один раз и существует до момента его уничтожения.

Покажем процедуру создания курсора, снабдив ее соответствующими комментариями.

```
/*Создание курсора*/
DECLARE PersCur CURSOR
    FOR SELECT Familia, DateRogd
    FROM Person
    WHERE Familia='Петров'
```

```
/*для объявленного курсора возможно использовать предложения OPEN, FETCH, CLOSE*/

OPEN PersCur

/*считывание содержимого полей в локальные переменные, как в хранимых процедурах */

FETCH PersCur INTO :Familia, :DateRogd

/*следует отметить, что локальные переменные могли быть заданы и при определении курсора */

DECLARE PersCur CURSOR
FOR SELECT Familia, DateRogd
INTO :Familia, :DateRogd
FROM Person
WHERE Familia='Петров'

/*в этом случае оператор извлечения данных меняется */

FETCH PersCur

/*не используемый курсор закрывается */

CLOSE PersCur
```

Результаты запроса могут считываться и в индексированные переменные

```
FETCH PersCur INTO :a('Результат запроса 1')
```

в данном случае — в трехуровневую переменную; в объекты, например объект с OREF=provider

```
FETCH PersCur INTO :Familia
Set provider.Familia=Familia.
```

Данные могут быть вставлены в курсор (запрос оператором INSERT) из переменных и индексированных переменных (как параметров).

1. Обращение к полям-спискам рассматривалось ранее.
2. Соединения могут быть внешними и неявными.

Во внешнем соединении (в отличие от внутреннего) «соединяемые» поля первой таблицы отражаются даже в том случае, если для поля во второй таблице не найдены соответствующие строки.

Во внешнем соединении CACHE в предложении WHERE вместо символа = применяется символ =*.

Неявное соединение определяется не запросом пользователя, а поддерживается БД. В неявных соединениях выделяют ссылки и отношения зависимости.

В *ссылке* поле ссылающейся таблицы содержит первичный ключ ID записи таблицы (указатель), на которую она ссылается.

```
SELECT Name, Manufacturer -> Surname
FROM Tovar
WHERE Type='инструмент',
```

что соответствует

```
SELECT Tovar.Name, Manufacturer.Surname
FROM Tovar, Manufacturer
WHERE Type="инструмент"
AND Tovar.Manufacturer=Manufacturer.ID.
```

Отношение зависимости — отношение 1 : M от родительской к дочерней таблице. Каждая строка дочерней таблицы ссылается на какую-либо строку родительской таблицы (вариант внутреннего соединения). Это чаще всего относится к встроенным объектам: пусть родительская таблица имеет имя Invoice, дочерняя — Position, а ссылка на нее — Invoice_Position.

Выделяют два типа зависимостей: от дочерней таблицы к родительской и от родительской к дочерней.

В первом случае оператор

```
SELECT Invoice->Date
FROM Invoice_Position
WHERE Price > 100000.
```

аналогичен

```
SELECT Invoice.Date
FROM Invoice_Position, Invoice
WHERE Price > 100000
AND Invoice_Position.Invoice=Invoice.ID.
```

Во втором случае оператору

```
SELECT Invoice_Position -> Price
FROM Invoice
WHERE Price > 100000
AND InvoiceNumber=1003274
```

соответствует

```
SELECT Invoice_Position_>Price
FROM Invoice, Invoice_Position
WHERE Invoice.InvoiceNumber=1003274
AND Invoice.ID=Invoice_Position.Invoice.
```

Во *встроенном языке* SQL-операторы «встраиваются» в программу на языке CACHE ObjectScript с помощью препроцессорной функции &sql, например

```
&sql(SELECT ID INTO :ID
FROM PERSON
WHERE Familia=:Familia)
Quit ..%OpenId(ID).
```

В качестве SQL-оператора могут выступать операторы обновления, создания объекта (CREATE) и описанный ранее CURSOR.

SQL-данные могут иметь следующие форматы: Logical (по умолчанию), ODBC, Display (не относится к свойствам-константам и значениям переменных).

Во встроенном языке SQL могут использоваться и макровызовы, для которых приняты обозначения: # — препроцессор, ## или & — расширение программного кода, $$$ — макровызов по сделанному ранее макроопределению.

Например,

```
#define TABLE Person
#define FIELDS Familia, Telephon
#define VARS :Familia, :Telephon
#define COND Familia %STARTWITH "A"

&sql(SELECT $$$FIELDS
INTO $$$VARS
FROM $$$TABLE
WHERE $$$COND).
```

С другой стороны, макроопределение может применяться для добавления &sql:

```
#define GETNEXT &sql(FETCH xcur INTO :a)
...
FOR $$$GETNEXT Quit :SQLCODE=100 Do abc,
```

что эквивалентно

```
FOR &sql(FETCH xcur INTO :a)
Quit :SQLCODE=100 Do abc.
```

Здесь SQLCODE — переменная, характеризующая выполнение &sql: 0 — успешно завершено; 100 — успешно завершено, но не найдено для заданных условий ни одной записи; менее нуля — имеется ошибка.

В заключение отметим, что наиболее удобно осуществлять SQL-доступ из внешних систем, что позволяет использовать такие реляционные объекты, как отчеты. Этот внешний реляционный доступ обеспечивается сервером CACHE SQL и интерфейсом ODBC.

7.6. ПЕРСПЕКТИВЫ РАЗВИТИЯ ООБД

По сравнению с реляционными БД ООБД обладают следующими преимуществами.

1. Лучшие возможности моделирования систем из блоков, обладающих произвольными связями.

2. Легкая расширяемость структуры за счет создания новых типов данных (свойств), наследования, установления новых связей и корректировки методов.

3. Возможность использования рекурсивных методов при навигационном методе доступа к большим объемам данных.

4. Повышение производительности в 10—30 раз.

5. Более широкая сфера применения (например, использование в мультимедийных системах).

Преимущества ООБД [3] приведут, видимо, к очень широкому их распространению. Однако прежде следует решить ряд задач по устранению недостатков ООБД: создать гибкую структуру БД; построить четкий язык программирования; отработать синтаксис разбора запросов, в том числе — вложенных; определить несколько методов доступа к данным; отработать вопросы одновременного доступа (разрешение конфликтов при множественном наследии); опреде-

лить сложный перебор данных; отработать защиту и восстановление данных; уточнить семантику (действия) операторов при динамических изменениях; встроить изменение атрибутов дочерних объектов.

Однако и после устранения названных недостатков переход к ООБД будет носить, видимо, эволюционный характер, поскольку сразу отказаться от значительного количества действующих реляционных БД будет нельзя. Такой безболезненный переход будет возможен, если первоначально в ООСУБД будет присутствовать не только объектная, но и реляционная составляющая. Более того, в ООСУБД следует ввести и многомерную модель для формирования хранилищ данных, парадигма которых хорошо согласуется с парадигмой ООБД. Именно такой подход использован в ООСУБД CACHE [38, 40].

Контрольные вопросы

1. Перечислите недостатки реляционных БД.
2. Что такое объектно-ориентированное проектирование? объектно-ориентированное программирование?
3. Что такое инкапсуляция? наследование? полиморфизм?
4. Что такое свойство, метод, событие?
5. Каковы тенденции развития ООБД?
6. Назовите основные типы (марки) ООБД.
7. Чем объектно-ориентированная БД отличается от объектно-реляционной БД?
8. Перечислите проблемы, которые еще следует решить в ООБД.
9. Назовите разновидности многомерной модели данных.
10. Что такое «многомерный куб»? Достоинства и недостатки MOLAP.
11. Объясните суть ROLAP. В чем отличие схем «звезда» и «снежинка»?
12. Укажите средние размеры хранилища данных.
13. Что такое «киоск (магазин, витрина) данных»?

Глава 8

Объектно-реляционная база данных

Переход к объектно-ориентированным моделям данных связан с процессом «перекачки» в них огромных объемов информации, которая в настоящее время хранится преимущественно в реляционных базах данных. Чтобы упростить этот процесс, сформировали объектно-реляционную модель данных, в которой выделяют две разновидности — гибридные и расширенные.

В гибридных объектно-реляционных базах данных объектно-ориентированный подход используется в создании интерфейса пользователя и алгоритма приложения. В то же время система таблиц формируется в рамках реляционной модели данных.

В расширенных объектно-реляционных базах данных объектно-ориентированный подход используется прежде всего при построении системы таблиц. Для этого разработана модификация языка SQL2, получившая название языка программирования SQL3.

8.1. ВИДЫ СТРУКТУР

«Промежуточной» моделью данных между реляционными и объектно-ориентированными базами данных является объектно-реляционная модель (ОРБД).

Ее появление вызвано двумя причинами.

1. Сложностью построения новой модели данных «с листа». Удобнее это делать на основе имеющихся проверенных разработок.

2. Учет преемственности с широко используемыми реляционными моделями, которые нельзя мгновенно заменить на объектно-ориентированные БД.

Различают, как отмечалось ранее, две разновидности ОРБД — гибридные и расширенные.

1. В **гибридных** ОРБД [1—3] интерфейс пользователя и алгоритм приложения выполнены с учетом объектно-ориентированного подхода, тогда как собственно БД является реляционной. Примерами могут служить СУБД Paradox и InterBase в рамках программного продукта Delphi. В каком-то смысле гибридной можно считать СУБД

Access при использовании языка программирования Visual Basic for Application (VBA).

2. В **расширенных (постреляционных)** ОРБД предполагается объектно-ориентированное построение собственно базы данных путем использования известных и введения новых типов данных, связанных между собой. Эта связь чаще всего осуществляется созданием методов с помощью триггеров и хранимых процедур. В расширенной объектно-реляционной модели [1—3] допускается, в отличие от реляционной модели данных, неатомарность данных в поле. В таких полях может располагаться другая таблица или массив. К подобным СУБД относятся Informix Universal Server, Oracle 8, UniSQL. В таких СУБД широко используется язык SQL3.

Заметим, что постреляционными [1] называют БД, в которых частично проведена денормализация данных, при этом допускается нетомарность записей.

Рассмотрим более подробно обе разновидности.

8.2. ГИБРИДНЫЕ ОРБД

Эта разновидность ОРБД рассмотрена на примере программного продукта Delphi.

Приложение Delphi изначально предназначалось для автоматизации программирования (объектно-ориентированного программирования), поскольку позволяло резко поднять производительность труда программистов за счет использования готовых «кубиков-программ». Требовалось лишь должным образом соединить эти «кубики».

В дальнейшем выяснилось, что примененный в Delphi объектно-ориентированный подход может быть использован при проектировании баз данных. В рамках Delphi применение объектно-ориентированного проектирования (см. рис. 7.2) и программирования не вызывает затруднений в интерфейсе пользователя и алгоритме приложения. Покажем это.

При работе Delphi на экране монитора появляется «картинка» (рис. 8.1). Компоненты-классы разделены на группы, определяемые соответствующими закладками (страницами).

Форма Delphii, сама являясь объектом, служит «коллектором» для объектов. Как только компонент помещается в форму (и получает порядковый номер), он становится объектом.

Заметим, что компоненты TPanel, TBevel также являются контейнерами (в форме), используемыми для форматирования, дизайна (разделения объектов и их выравнивания). Другими «миниконтейнерами» служат компоненты DataModule и TQuickRep (отчет).

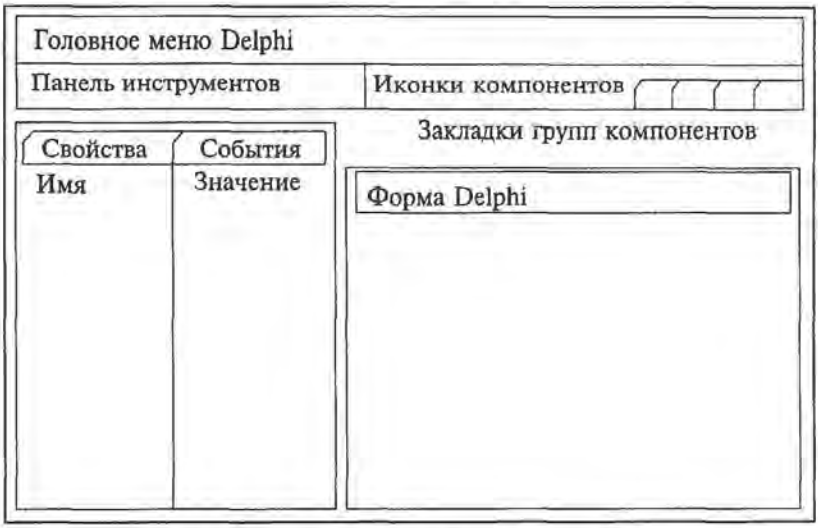


Рис. 8.1. Экран для работы с приложением Delphi

Для активного объекта в форме при построении программ в «Инспекторе объектов» могут использоваться свойства (страница «свойства») и события (закладка «события»), обычно запускающие программы-методы.

Все компоненты разделены на следующие основные группы: Standard, Additional, Dialogs, Win32, System, VCL, Internet, DataAccess, DataControl, QReport, Decision Cube, ActiveX. В процессе автоматизированного программирования чаще всего используют первые четыре страницы.

С позиций собственно баз знаний и баз данных следует обратить особое внимание на страницы DataAccess и DataControl. К ним примыкают страницы QReport, Decision Cube.

Состав компонентов некоторых закладок DataAccess и DataControl показан на рис 8.2. На рис. 8.3 и 8.4 показаны свойства и события компонента TTable.

Рассмотрим описание программных возможностей Delphi.

Приложение Delphi может работать с тремя реляционными СУБД: dBase, Paradox — в локальном режиме (рис. 8.3); InterBase, который инсталлируется отдельно, — в режиме клиент—сервер.

Заметим, что СУБД InterBase возможно использовать в двух вариантах: *локальном* (сервер и клиент на одном компьютере, используется обычно в процессе отладки) и *удаленном* (сервер и клиент на разных компьютерах).

Standard	Additional	DataAccess	DataControl	Dialog
MainMenu	BitMapButton	DataSource	DBGrid	OpenDialog
PopupMenu	SpeedButton	Table	DBNavigator	SaveDialog
Label	TabSet	Query	DBText	FontDialog
EditBox	MaskEdit	StoredProc	DBEdit	ColorDialog
Memo	OutLine	DataBase	DBCheckBox	PrintDialog
Button	Shape	BatchMove	DBRadioGroup	FindDialog
CheckBox	Bevel	Session	DBComboBox	ReplaceDialog
RadioButton	ScrollBox	UpdateSQL	DBRichEdit	
ListBox			DBListBox	
ComboBox				
ScrollBar				
GroupBox				
RadioGrupp				
Panel				

Рис. 8.2. Состав некоторых страниц компонента Delphi

Независимо от используемой СУБД интерфейс пользователя и алгоритм приложения строится с использованием объектно-ориентированного подхода. Подтвердим кратко это утверждение.

А. Речь первоначально пойдет о реализации **собственно базы данных**.

А1. Первоначально задается имя БД. С помощью этого имени осуществляется ссылка на БД. Однако при этом требуется указывать порой длинный путь (адрес) доступа, что неудобно при многократном обращении к БД. В силу этого чаще используется *алиас* (*синоним*) БД — имя, заменяющее длинный путь. Построение алиаса ведется с помощью Delphi-утилит BDE Administrator и SQL Explorer.

А2. Создание таблиц БД для СУБД Paradox (локальный режим) и InterBase (режим «клиент—сервер»), равно как и заполнение БД данными, специфично для каждого случая. Нетрудно видеть, что при создании собственно базы данных не используется объектно-ориентированный подход.

Б. Интерфейс. Для построения интерфейса нет четких формальных правил. Однако некоторые рекомендации могут быть сформулированы.

Интерфейс составляется с учетом пожеланий пользователя. Вид интерфейса зависит от инструментария, предоставляемого СУБД или программным приложением.

В качестве инструментов для интерфейса могут быть система форм (Form), система кнопок, система элементов управления (список, поле со списком и т. д.).

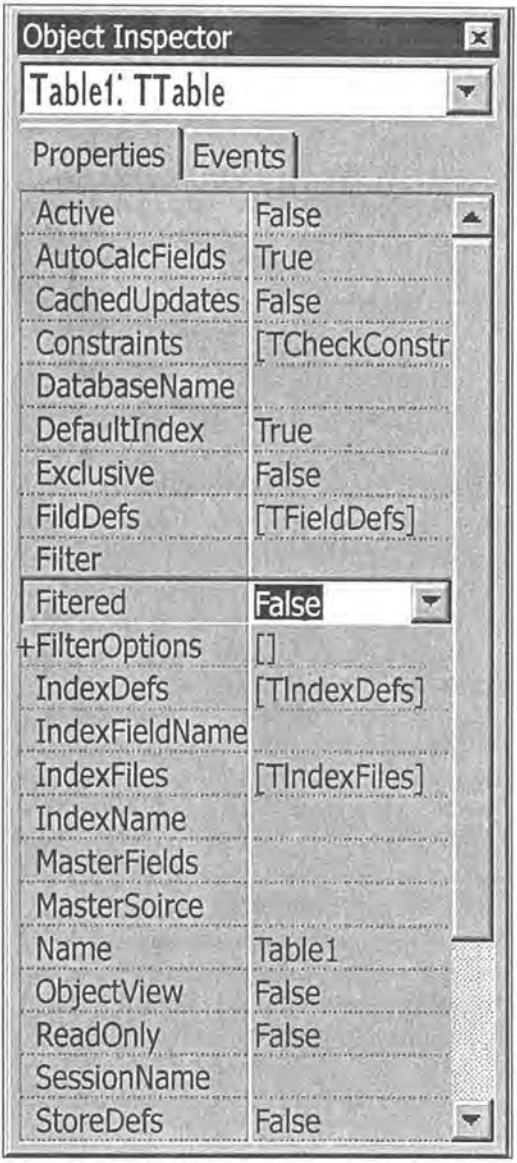


Рис. 8.3. Свойства компонента Table

Для Delphi исходным инструментом чаще является система форм, вызываемая из головного меню или с помощью системы элементов управления и меню в главной форме.

Б1. Формы Delphi.

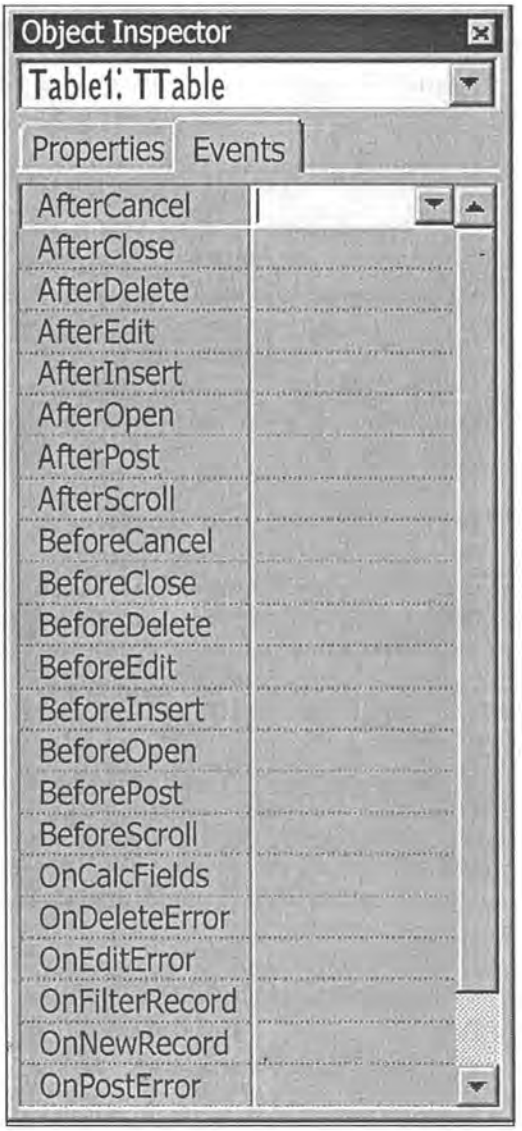


Рис. 8.4. События компонента Table

Б1.1. Формы Delphi часто называют окнами. Окно называют *модальным* (М), если доступ к предыдущему открытому окну возможен только после закрытия М-окна. В противном случае окно называют *немодальным*.

Пусть имеются две формы — Form1 и Form2, при этом Form2 вызывается из Form1.

В случае немодального окна для формы Form1 пишется программный модуль unit1

```
procedure TForm1.Button1Click(Senser: Tobject);
begin
Form2.Show;
end;
```

Чтобы можно было работать со вторым окном (формой), в программе unit1 (раздел implementation) добавляется ссылка на программный модуль unit2 (формы Form2)

```
implementation
uses unit2;
```

Теперь из unit1 можно запускать немодальное окно Form2.

Чтобы сделать Form2 модальным окном, следует установить ее свойство BorderStyle=bsDialog. При этом надо внести изменения в unit1: вместо Form1.Show написать Form2.ShowModal.

Б1.2. Вместо «ручного» создания многооконного интерфейса возможно использовать шаблоны, в которых выделяются два варианта:

- многодокументный интерфейс (Multiple Document Interface — MDI);
- однодокументный интерфейс (Simple Document Interface — SDI).

При использовании MDI дочерние окна не превышают по размерам родительское окно и могут располагаться мозаикой (Title) или каскадом (Cascade). В родительском окне содержится главное меню приложения. MDI организуется так же, как модальное окно, при этом свойство FormStyle=fsMDIForm для родительского окна и FormStyle=fsMDIChild — для дочерних окон. При создании MDI возможно использование шаблона или построение MDI с «нуля».

В последнем случае выбирается File/New головного меню Delphi и в диалоговом окне New Items на странице New выделяют значок Application, вводя по очереди родительское и дочерние окна.

SDI может содержать также несколько окон, при этом дочерние окна не ограничены размерами родительского окна, называемого иногда главным.

В программный модуль unit главного окна включаются ссылки на все дочерние unit-модули:

```
implementation
uses SDIWdw;
```

В unit-программах дочерних окон должны быть ссылки на unit-программу главного окна:

```
implementation
uses SDIMain;
```

Перед сворачиванием главного окна первоначально сворачиваются дочерние окна.

Лучше использовать шаблоны SDI (MDI). Для этого обращаются к головному меню Delphi (File/New), входят в диалоговое окно New Items и на вкладке (странице) Projects выбирают необходимый шаблон: SDI (одна форма и один программный модуль) или MDI (две формы и три модуля).

Б2. Последовательность операций работы пользователя может быть задана с помощью меню или системы кнопок.

Б2.1. Формирование меню достаточно просто и потому подробно здесь не рассматривается. Упорядочение элементов меню (фактически — объектов), написание программ для них не отличаются от «привязывания» программ к таким элементам управления, как кнопка, поле со списком и т. д. Заметим лишь, что есть две разновидности меню — главное (TMainMenu страницы Standard) и всплывающее меню (TPopupMenu), вызываемое нажатием правой кнопки мыши.

Б2.2. Другой возможностью задания последовательности операций работы пользователя является использование системы кнопок (TButton, TBitBtn). В Delphi обычно используют меню, а кнопки чаще играют вспомогательную роль и используются, например, для закрытия форм, изменения доступа к элементам меню.

Свойствами меню, равно как и системы кнопок, наиболее часто используемыми, являются Enabled и Visible.

Б3. Гибкость меню обеспечивается использованием элементов управления.

Б3.1. Универсальные элементы управления размещены на страницах Standard, Additional, Dialog палитры компонент. К таким элементам относятся TLabel, TEdit, TMemo, TComboBox. TButton, TBitBtn.

Специфично использование TLabel. Свойство Caption применяют для задания заголовков таблиц БД, названий к таким компонентам, как TEdit, TMemo.

В свою очередь свойства Edit1.Text и Memo1.Line применяют для задания параметров (одно- и многострочных), например, в запросах. Пожалуй чаще всего перечисленные элементы управления используют свойства Enabled, Visible.

Б3.2. Специализированными элементами управления, связанными непосредственно с БД, являются компоненты страницы DataControl. К ним следует отнести прежде всего DBGrid, DBEEdit, DBNavigator, DBMemo.

Компоненты DBEdit позволяют создать форму базы данных в столбец. Такие формы часто применяются при работе в СУБД Access для заполнения БД.

Б4. Сообщение. Его можно сделать обычным, используя, например,

```
MessageDlg(Query1['Name'], mtInformation, [mbOk], 0);
```

Однако предпочтительнее применить схему исключительной ситуации try ... except

```
begin
try
Query1['Plan']< Query1['Fakt']
except

ShowMessage('Число вакансий меньше числа принимаемых. Измените правила или должности принимаемых');

Exit;
end;
```

Программа сработает, если есть исключительная ситуация.

Интерфейс пользователя образуется и конечным объектом описываемых далее «цепочек» доступа. Осветим их в алгоритме приложения (АП), поскольку остальные объекты «цепочек» участвуют именно в АП.

В. Алгоритм приложения. Программы приложений могут быть написаны на одном или сочетании трех языков: интерфейсный SQL, вложенный *SQL*, *Object Pascal* (*OP*) .

В.1. Возможны три «цепочки» доступа (рис. 8.5):

TTable — *TDataSource* — *TDBGrid* (чаще используется в СУБД Paradox);

TQuery — *TDataSource* — *TDBGrid* (чаще применяются для СУБД InterBase);

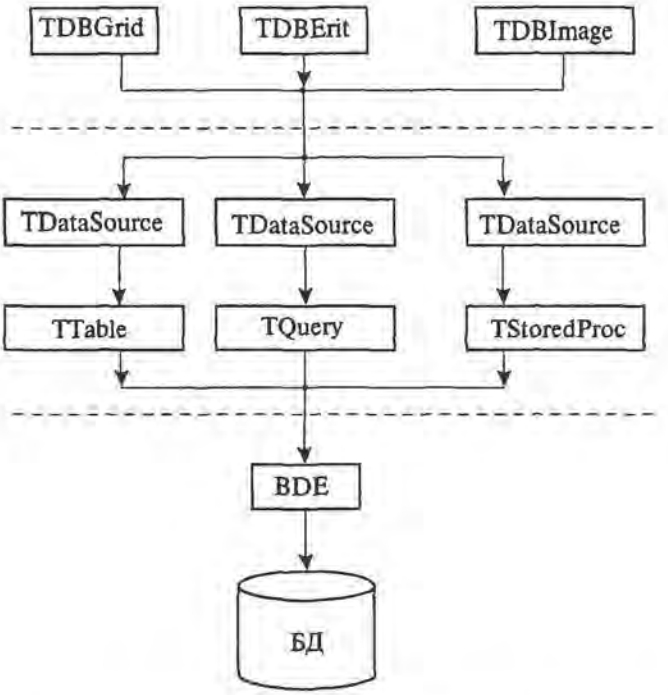


Рис. 8.5. Объектно-ориентированный интерфейс пользователя (*приложение Delphi*):

BDE — Borland Database Engine (ядро Delphi)

TStoredProc — TDataSource — TDBGrid (используют только для СУБД InterBase).

Будем их далее называть «цепочками» Table, Query и StoredProc соответственно.

В этих «цепочках» компонент TDBGrid является визуальным, а остальные — невизуальны. Компоненты TTable, TQuery, TStoredProc обеспечивают непосредственный доступ к БД (по алиасу), тогда как компонента TDataSource играет роль «размножителя»: к нему могут подключаться визуальные компоненты TDBNavigator, TDBText, TDBEdit, TDBMemo, TDBListBox, TDBComboBox и другие. Эти подключаемые компоненты (если «спрятать» компонент TDBGrid с помощью свойства Visible или, лучше, Enabled) могут образовывать «аналог» форм Access (для заполнения БД). Понятие «форма» в смысле Access в Delphi отсутствует.

Связи в «цепочках» устанавливаются в статическом режиме (построения доступа), либо в динамическом при выполнении программы.

«Цепочка» StoredProc специфична и используется только в режиме клиент—сервер, да и то редко.

В.2. Наряду с перечисленными «цепочками» могут использоваться и «укороченные цепочки», составленные из невизуальных объектов, контейнером для которых служит модуль (объектов) DataModule.

Такой контейнер организуется через меню Delphi (File/NewDataModule). Связь между компонентами TTable — TDataSource и TQuery — TDataSource устанавливается так же, как это делается в форме при использовании ее в качестве контейнера для рассмотренных ранее «цепочек».

DataModule сохраняется под каким-либо именем, и для него создается программный модуль unit, имя которого добавляется в тексты программных модулей других форм приложения (File/Use Unit). При ссылках на такой модуль (объектов) следует сначала указать его имя, например DataModule1.DataSource1.

До сих пор речь шла о доступе к данным без их изменения. Однако при работе с БД имеют место операции обновления (вставить, изменить, уничтожить). Они могут выполняться вручную или автоматически (при выполнении программ).

В первом случае это имеет место при создании БД (заполнении БД данными). Обновление может осуществляться непосредственно в таблицах или через своеобразные формы, составленные из компонент TDBEdit. Такие обновления не вызывают затруднений.

В3. Вложенный SQL используется только в режиме клиент—сервер.

В4. Интерфейсный SQL используется в локальном режиме и режиме клиент—сервер.

Заметим, что интерфейсный язык SQL достаточно гибок при работе с БД, однако в нем нет понятия циклов и переходов. Кроме того, в свойстве SQL компоненты TQuery (или свойстве любой компоненты, например Memo.Text, из которой заимствуется SQL-оператор) может быть записан и запущен *только один оператор*, после выполнения которого его следует удалить и ввести новый. Таким образом, для выполнения нескольких операторов их следует вводить по одному, для чего требуется непростое программное решение, или «встраивать» в операторы Object Pascal.

В силу сказанного сфера действия интерфейсного языка SQL ограничена. Его операторы в чистом виде чаще всего используют в единичных запросах как в локальном режиме, так и в режиме клиент—сервер.

В локальном режиме чаще используют язык Object Pascal, составляющими операторами которого могут быть SQL-операторы.

В5. Основную роль в локальном режиме играет язык программирования Object Pascal. В режиме клиент—сервер он, совместно с интерфейсным языком SQL, используется в клиентской части.

Следует отметить, что в Object Pascal существуют две разновидности программ:

1) «привязанные» непосредственно к событиям;

2) не связанные напрямую с событием (обычно «привязанные к форме») и вызываемые из других программ.

Построение первых программ широко описано в литературе, тогда как о вторых практически не упоминается. Последние пишутся полностью вручную.

С помощью реляционных БД и данной разновидности ОРБД пытались решить задачу создания базы данных для хранения графических данных и данных с большим объемом полей. При этом сами данные хранились в файлах, а в базе данных имелись лишь ссылки на эти файлы. Однако такой путь оказался непродуктивным из-за возникающих многочисленных проблем.

Таким образом, в гибридной ОРБД интерфейс пользователя и алгоритм приложения построены с использованием объектно-ориентированного подхода, а собственно база данных — реляционная.

8.3. РАСШИРЕННЫЕ ОРБД

К расширенным ОРБД относится СУБД Informix Universal Server [41, 42]. С помощью этой разновидности ОРБД решают две группы задач: 1) хранение в БД данных большого объема; 2) трансформация реляционной модели (собственно БД) в объектно-реляционную модель данных.

Обсудим состояние решения этих задач.

Задача 1. Первоначально графические базы данных создавались так: отдельно формировались графические файлы, а в столбцах СУБД были лишь ссылки на эти файлы. Это требовало от программиста знания не только БД, но и системы файлов. В связи с этим в состав СУБД были первоначально введены «простые большие объекты» (рис. 8.6) TEXT и BYTE с объемом до 1 Гбайт.

Вскоре выяснилось, что и такая размерность поля порой недостаточна, и были созданы интеллектуальные большие объекты Binary Large Object (BLOB) и CLOB размером до 4 Тбайт. Работа с такими объектами возможна только «по частям», для чего пришлось создать специальные дополнительные технологии.

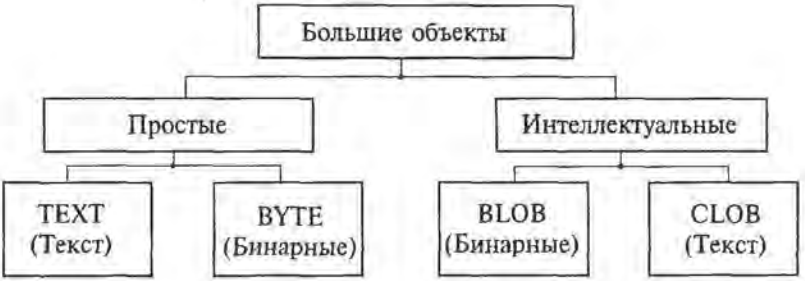


Рис. 8.6. Большие объекты

Задача 2. Идею расширения (системы данных) удобно иллюстрировать на примере [41] программы (среда Delphi, язык Object Pascal):

```
unit polimor_;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
    StdCtrls;

type
    TForm1 = class(TForm)
        Edit1: TEdit;
        Edit2: TEdit;
        GroupBox1: TGroupBox;
        RadioButton1: TRadioButton;
        RadioButton2: TRadioButton;
        Label1: TLabel;
        Label2: TLabel;
        Button1: TButton;
        Button2: TButton;
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
    private
        { Private declarations }
    public
        {Public declarations }
    end;
```

```
    type
        TPerson=class
        fname:string;{ имя }
        constructor Create(name:string);
        function info:string; virtual;
    end;
    TStud=class(TPerson)
        fgr:integer; { номер группы }
        constructor Create(name:string;gr:integer);
        function info:string; override;
    end;
    TProf=class(TPerson)
        fdep:string; { название кафедры }
        constructor Create(name:string;dep:string);
        function info:string; override;
    end;
    const
        SZL=10; // размер списка
    var
        Form1: TForm1;
        List: array[1..SZL] of TPerson; // список
        n:integer;                  // кол-во людей в списке
implementation
{$R *.DFM}
constructor TPerson.Create(name:string);
        begin
        fname:=name;
    end;
constructor TStud.Create(name:string;gr:integer);
    begin
        inherited create(name);
        fgr:=gr;
    end;
constructor TProf.create(name:string; dep:string);
    begin
        inherited create(name);
        fdep:=dep;
    end;
```

```
function TPerson.Info:string;
    begin
        result:=fname;
    end;

function TStud.Info:string;
    begin
        result:=fname+' гр.'+IntToStr(fgr);
    end;

function TProf.Info:string;
    begin
        result:=fname+' каф.'+fdep;
    end;

procedure TForm1.Button1Click(Sender: TObject);
  begin
    if n<=SZL
    then
      begin
        if Radiobutton1.Checked
          then // создадим объект TStud
            List[n]:=TStud.Create(Edit1.Text,StrToInt(Edit2.Text))
          else
        List[n]:=TProf.Create(Edit1.Text,Edit2.Text);
      n:=n+1;
    end
  else ShowMessage('Список заполнен!');
end;

procedure TForm1.Button2Click(Sender: TObject);
var
    i:integer;
    st:string;
begin
    for i:=1 to SZL do
        if list[i] <> NIL then st:=st+list[i].info+#13;
    ShowMessage('Список'+#13+st);
end;

end.
```

Из нее видно, что классы «Студент» (TStud) и «Профессор» (TProf) являются производными от класса «Личность» (TPerson). В качестве

методов доступа к ним выступают функции TStud.Info.string, TProf.Info.string и TPerson.Info.string соответственно.

Язык программирования Object Pascal редко используется для формирования объектно-ориентированной модели базы данных. Чаще для этих целей применяют языки C++ и SQL.

Покажем на примерах использование языка SQL3 в СУБД Informix Universal Server.

В SQL3 используются те же типы данных, что и в SQL2. Однако SQL3 оперирует не таблицами, как SQL2, а объектами. В качестве таких объектов выступают новые (абстрактные) типы данных и система таблиц, образующих иерархию.

А. Абстрактные типы данных (рис. 8.7). Абстрактные типы данных, получившие в последнее время название «типы данных, определяемые пользователем» [2], — типы, задающие характеристики [3], но не реализации, наследуемые подтипом. Они непосредственно не порождают экземпляров.

В иерархии типов участвуют (рис. 8.7) ROW (неименованные строки), ROW TYPE (именованные строки), SET (неупорядоченный набор одного типа без повторения данных), MULTISET (то же, что и SET, с возможностью повторения данных), LIST (упорядоченный набор неуникальных данных одного типа, фактически — одномерный массив). Разнородные структуры добавляют каждый раз по одной строке к предыдущей строке, тогда как однородные структуры могут добавлять по нескольку строк к каждой предыдущей строке.

Использование новых типов рассмотрим на системе таблиц, показанных на рис. 8.8. Иерархия требует множественного наследования. Поскольку оно не реализовано в рамках СУБД Informix Universal Server, ограничимся рассмотрением части иерархии, обведенной на рис. 8.8 пунктиром.

Речь должна пойти о построении типов данных, их заполнении и обновлении. Первоначально покажем использование типа данных ROW для элементов иерархии «Деканат», «Группа», «Студент», «Ка-

Рис. 8.7. Новые абстрактные типы данных

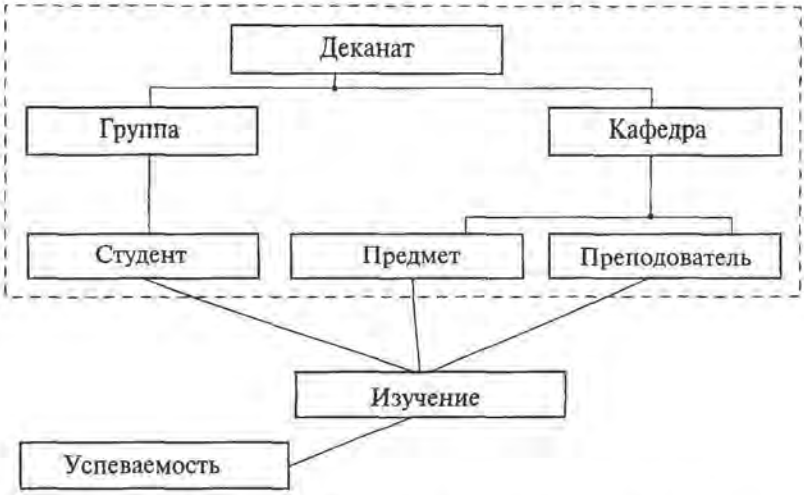


Рис. 8.8. Иерархия абстрактных типов данных и таблиц

федра». Для иллюстрации в каждом структурном элементе используем только по два поля, а названия таблиц и полей выполним на русском языке.

```
CREATE ROW TABLE Деканат(
Шифр_деканата integer,
Название varchar(15),
    Группа ROW(
    Шифр_группы varchar(8),
    Название_группы varchar(5),
        Студент ROW(
        Шифр_студента varchar(8),
        Фамилия varchar(15)),
    Кафедра ROW(
    Шифр_кафедры integer,
    Название_кафедры varchar(50))
    );
```

Выборка данных осуществляется следующим образом.

```
SELECT шифр_деканата, группа.название_группы, группа.студент.фамилия
    FROM Деканат
        WHERE группа.название_группы='И5';
```

Заполнение (вставка) в такую структуру проводится так:

```
INSERT INTO Деканат
    VALUES (1, 'ПТИО',
        ROW('И-99', 'И4',
            ROW('И-99/15', 'Петров')),
        ROW(1, 'ИиУС'));
```

В типе данных ROW структура строки (записи) каждый раз должна определяться отдельно. Это не позволяет использовать структуру ROW многократно.

Такую возможность предоставляет именованная запись ROW TYPE. В этом случае предыдущая структура «Деканат» задается иначе.

Первоначально формируются именованные записи.

```
CREATE ROW TYPE Студент_TYPE(
        Шифр_студента varchar(8),
        Фамилия varchar(15)
);
CREATE ROW TYPE Группа_TYPE(
    Шифр_группы varchar(8),
    Название_группы varchar(5),
    Студент Студент_TYPE
);
CREATE ROW TYPE Кафедра_TYPE(
    Шифр_кафедры integer,
    Название_кафедры varchar(50)
);
```

Из именованных записей составляется структура «Деканат»

```
CREATE ROW TABLE Деканат(
Шифр_деканата integer,
Название varchar(15),
Группа Группа_TYPE,
Кафедра Кафедра_TYPE
);
```

При этом сформируется таблица с нелинейной структурой (табл. 8.1).

Таблица 8.1

Сформированная таблица «Учебный процесс»

Шифр деканата	Название деканата	Группа		Студенты		Кафедра	
		Шифр группы	Название группы	Шифр студента	Фамилия	Шифр кафедры	Название кафедры
1	ПТиО	И-99	И4	И-99/12	Петров	1	ИиУС
1	ПТиО	И-99	И4	И-99/11	Смирнов	1	ИиУС
...	...	...	...	...	...	...	...

Можно таблицу построить и другим способом.

```
CREATE ROW TYPE Деканат_TYPE(
Шифр_деканата integer,
Название varchar(15),
Группа Группа_TYPE,
Кафедра Кафедра_TYPE
);
```

и

```
CREATE TABLE Деканат OF TYPE Деканат_TYPE;
```

Последняя таблица получила название *типизированной*. Заметим, что при вставке в тип ROW проблем не возникает:

```
INSERT INTO Деканат
    VALUES (1, 'ПТИО',
        ROW('И-99', 'И4',
            ROW('И-99/15',
'Петров')::Студент_TYPE)::Группа_Type,
        ROW(1, 'ИиУС')::Кафедра_TYPE);
```

где знак :: обозначает перевод записи в элемент ROW.

Операторы заполнения ROW TYPE сложнее, чем ROW, и требуют написания хранимых процедур или функций. Например, список групп в деканате формируется следующим образом:

```
CREATE FUNCTION Список_групп(Деканат.ГруппаTYPE)
    RETURNS(varchar(8), varchar(5)) AS
    RETURN(Группа.Шифр_группы, Группа.Название_группы)
END FUNCTION;
```

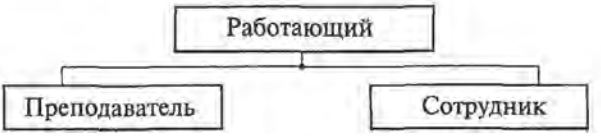


Рис. 8.9. Иерархия абстрактных типов данных (наследование)

Таблицу «Преподаватель» на рис. 8.8 временно заменим на таблицу «Работающий», подразумевая, что преподаватель одновременно и преподаватель, и (научный) сотрудник (рис. 8.9).

```
CREATE ROW TYPE адрес_TYPE(
        Город varchar(15),
        Улица varchar(15),
        Дом integer,
        Корпус integer,
        Квартира integer
);

CREATE ROW TYPE Работающий_TYPE(
        Шифр_работающего integer,
        Фамилия varchar(15),
        Адрес адрес_TYPE
);
```

В составе «Работающий» имеются подчиненные ему элементы «Преподаватель» и «Сотрудник». Тогда иерархия подчинения (наследования) типов данных формируется так:

```
CREATE ROW TYPE Преподаватель_TYPE(
    Дисциплина varchar(20),
    Вид_занятий varchar(10),
    UNDER Работающий_TYPE;

CREATE ROW TYPE Сотрудник_TYPE(
    Шифр_работы varchar(5),
    Название работы varchar(40),
    UNDER Работающий_TYPE;
```

Тогда типизированные таблицы имеют вид:

```
CREATE TABLE Преподаватель OF TYPE Преподаватель_TYPE;
```

```
CREATE TABLE Сотрудник OF TYPE Сотрудник_TYPE;
```

Два последних определения могут включать и внешние ключи.

Фактически сформирована система таблиц. Она отличается тем, что заполнение таблиц идет автономно. Если заполняются поля в таблице «Сотрудник», то одноименные поля в таблице «Работающий» не заполняются.

Б. Иерархия таблиц. Указанное заполнение порой неудобно и потому используют наследование не абстрактных типов данных, а таблиц, которые связывают так:

```
CREATE TABLE Преподаватель
    OF TYPE Преподаватель_TYPE
    UNDER Работающий;

CREATE TABLE Сотрудник
    OF TYPE Сотрудник_TYPE
    UNDER Работающий;
```

В этом случае таблицы становятся вложенными (рис. 8.10).

Если создать запрос

```
SELECT * FROM Работающий;
```

то будут выданы поля как объекта «Работающий», так и объектов «Преподаватель» и «Сотрудник».

Для получения данных только из определенной таблицы (например, «Сотрудник») необходимо создать запрос

```
SELECT * FROM ONLY(Сотрудник);
```

Аналогично и с процедурами обновления

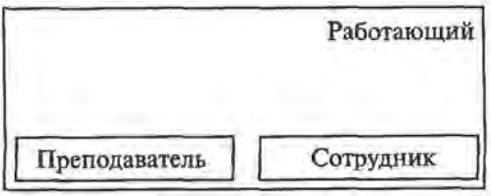


Рис. 8.10. Пример вложенной таблицы

```
DELETE FROM ONLY(Сотрудник)
WHERE Шифр_работы=254;
```

До сих пор использовались (см. рис. 8.6) разнородные структуры. Однако таблицы могут быть пополнены и однородными структурами (коллекциями).

Добавим к таблицам «Сотрудник» и «Преподаватель» коллекции:

а) участие сотрудника в научных темах:

```
ALTER TABLE Сотрудник
        ADD Проект SET (integer);
```

б) учебные пособия преподавателя:

```
ALTER TABLE Преподаватель
        ADD Пособия MULTISET (ROW(
        Название varchar(30),
        Год_издания integer)
);
```

Заметим, что MULTISET состоит из наборов ROW.

Тогда таблица «Преподаватель» получает новый вид (табл. 8.2).

Таблица 8.2

Новая таблица «Преподаватель»

Шифр работающего	Фамилия	Шифр предмета	Вид занятия	Адрес					Пособия	
				Город	Улица	Дом	Корпус	Квартира	Название	Год издания
28	Петров	1	Лекции	СПб	Кима	21	2	321	А Б В	1999 1998 1995

Отметим также, что коллекция — понятие гибкое. Она может быть полем записи и в то же время записи могут играть роль элементов коллекции.

Запросы к коллекциям могут быть составлены лишь в простейших случаях. Запрос на выборку для таблицы «Преподаватель» имеет вид

```
SELECT шифр_работающего, фамилия,
Пособия.Название
```

```
FROM Преподаватель
WHERE 1999 IN (Пособия);
```

Вставка может осуществляться таким оператором

```
INSERT INTO Преподаватель
VALUES (15, 'Иванов', 1, 'преподаватель',
ROW('СПб', 'Московский', 137, 2, 118),
ROW('Информатика', 'лекция'),
"MULTISET{ROW('A1',1996),
ROW('A2',1997),
ROW('A4',1997)}");
```

Следует отметить, что в СУБД Informix Universal Server введен новый тип переменной COLLECTION, в которой в процессе работы программы (хранимой процедуры) могут храниться значения данных.

Для более сложных запросов требуется использовать хранимые процедуры, возможно, с циклами.

С помощью хранимых процедур формируются и необходимые дополнительные методы, поскольку в СУБД Informix Universal Server нет механизма превращения хранимых процедур в методы. Таким образом, язык SQL3 обладает значительно большими возможностями по сравнению с языком SQL2.

В то же время разработчиков БД, видимо, настораживает более сложная структура объектов и, главное, методов. Кроме того, SQL3 характеризуется неполнотой операторов: часть операторов разработчику приходится создавать самому в рамках хранимых процедур, для чего необходимо хорошо знать язык SQL2.

Названные обстоятельства и инерционность привычного, широко распространенного реляционного мышления объясняют, по всей видимости, тот факт, что расширенные ОРБД не нашли пока масштабного применения.

Вместе с тем широкое распространение получила гибридная разновидность ОРБД, являющаяся серьезным усовершенствованием реляционных БД.

Расширенные ОРБД — серьезный шаг в направлении объектно-ориентированных баз данных. Сфера применения таких ОРБД будет, по мнению авторов, расти. Они формируют средства как для революционного перехода к ООБД, так и для постепенного, эволюционного движения в будущее баз данных.

8.4. ПЕРСПЕКТИВЫ РАЗВИТИЯ ОРБД

Чтобы понять перспективы развития ОРБД, следует оценить их достоинства и недостатки

К достоинствам следует отнести [2]:

- устранение ряда недостатков реляционных БД (см. § 8.1);
- повторное использование компонентов;
- использование накопленных знаний по реляционным БД.

К недостаткам ОРБД возможно отнести:

- усложнение структуры БД и частичную утрату простой обозримости результатов, как в реляционных БД;
- сложность построения абстрактных типов данных и методов, связывающих типы в иерархию;
- менее широкий набор типов связей, определяемых языком программирования SQL, чем в объектно-ориентированных БД;
- менее продуманный, отлаженный и стандартизованный набор типов данных, чем в ООБД.

ОРБД, по-видимому, будут существовать еще достаточно долго, чему есть по меньшей мере два объяснения.

1. Быстрое накопление с помощью ОРБД опыта, который можно использовать при создании ООСУБД.

2. Необходимость иметь средства для постепенного эволюционного «перевода» многочисленных реляционных БД в разряд ООБД, за которыми, видимо, будущее.

Контрольные вопросы

1. Назовите разновидности ОРБД. В чем их отличие?
2. В чем суть гибридной ОРБД? расширенной ОРБД?
3. Покажите место использования объектно-ориентированного подхода в обеих разновидностях ОРБД.
4. Какие СУБД используются в программном продукте Delphi?
5. Назовите достоинства и недостатки ОРБД.
6. Какие контейнеры имеются в Delphi?
7. Что такое класс, компонента, объект в объектно-ориентированном программировании?
8. Назовите задачи, решаемые расширенной реляционной БД.
9. Расскажите о назначении больших объектов.
10. Назовите новые абстрактные типы данных.
11. Как наследуются типы данных и таблицы? В чем отличие наследования?

Глава 9

Взаимосвязь моделей данных, физическая организация БД

Приводится сравнительная характеристика достоинств различных моделей данных, правила преобразования данных из одной модели в другую, что очень важно при построении распределенной базы данных из уже действующих локальных баз данных с одинаковыми или разными моделями данных.

Рассматривается процедура выбора модели данных (в общем случае — неоднозначная) в процессе проектирования базы данных.

Обсуждаются вопросы физического построения базы данных: методы размещения, обновления и доступа к данным в базе данных.

9.1. СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА МОДЕЛЕЙ ДАННЫХ, ПРЕОБРАЗОВАНИЕ МОДЕЛЕЙ ДАННЫХ

Описанные в гл. 5—8 свойства реляционных, сетевых, иерархических, объектно-иерархических и объектно-реляционных моделей данных (многомерные модели считаем разновидностью иерархических моделей данных) возможно систематизировать и сравнить (табл. 9.1). Из таблицы видно, что ни одна из моделей не удовлетворяет полностью требованиям, предъявляемым к современным БД.

В связи с этим возникает ряд проблем, основными из которых являются: 1) преобразование моделей данных; 2) выбор модели данных и СУБД.

Обсудим эти проблемы.

Детально процедуры преобразования (отображения) рассмотрены позднее, здесь рассмотрим общую постановку задачи.

Возможные варианты преобразований [10] МД (в себя и другие модели) показаны на рис. 9.1.

В преобразовании выделяют трансформацию структур (схем и ограничений), определяемых ЯОД (позиция А — операции не охва-

Таблица 9.1

Сравнительная характеристика моделей БД

Вид модели	Достоинства	Недостатки
Иерархическая	Простота понимания Высокое быстродействие при совпадении структур базы данных и запроса	Отношения M:M могут быть реализованы только искусственно Могут быть избыточные данные Усложнение операций включения и удаления Удаление исходных сегментов приводит к удалению порожденных сегментов Процедурный характер построения структуры БД и манипулирования данными Доступ к любому порожденному сегменту возможен только через корневой сегмент Сильная зависимость логической и физической моделей Ограниченный набор структур запроса Невозможность реализации таблиц с нелинейной структурой
Сетевая	Сохранение информации при уничтожении записи-владельца Более богатая структура запросов Меньшая зависимость логической и физической моделей Возможность реализации таблиц с нелинейной структурой	Отношения M:M могут быть реализованы только искусственно Необходимость программисту знать логическую структуру БД Процедурный характер построения структуры БД и манипулирования данными Возможная потеря независимости данных при реорганизации БД

Продолжение табл. 9.1

Вид модели	Достоинства	Недостатки
Реляционная	Произвольная структура запроса Простота работы и отражения представлений пользователя Отделение физической модели от логической и логической от концептуальной Хорошая теоретическая проработка	Отношения M : M могут быть реализованы только искусственно Необходимость нормализации данных Возможность логических ошибок при нормализации и реализации Невозможность реализации таблиц с нелинейной структурой
Объектно-ориентированная	Неограниченный набор типов данных Возможность реализации таблиц с нелинейной структурой Послойное представление данных Высокая скорость работы из-за отсутствия ключа Ненужность нормализации Легкая расширяемость структуры и ее гибкость Повторное использование типов данных и компонент Реализация отношений M : M	Сложность освоения модели из-за сложности структуры БД Нечеткий язык программирования Недостаточная защита данных Нечетко проработанный одновременный доступ Плохая обозримость структуры

чены), операций, соответствующих ЯМД (позиция Б). Следует заметить, что при ручном преобразовании МД, как это сделано в гл. 5—9, взаимооднозначного соответствия может (из-за ограничений) и не быть, поэтому на рис. 9.1 введено понятие изоморфизма — конструктивности (позиция Б).

Считаем, что отображения *конструктивны*, если реализация БД, соответствующая одной исходной схеме S_s, отображается в другую реализацию, соответствующую другой, целевой схеме S_t:

$$S_s \rightarrow S_t.$$

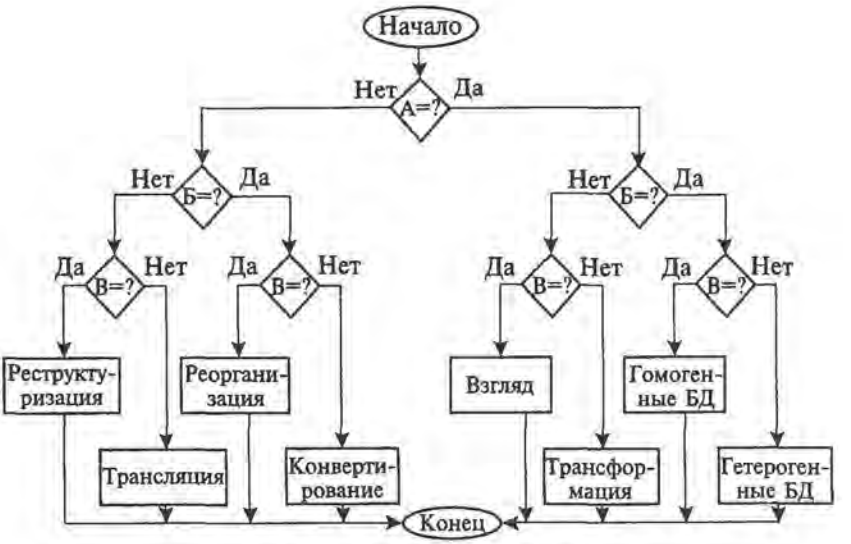


Рис. 9.1. Виды преобразований:
А — операции охвачены; Б — схемы изоморфны; В — модели одинаковы

В общем случае это отображение — гомоморфизм. Кратко опишем возможные формы преобразования.

Реструктуризация — изменение структуры в рамках одной МД: схема отношения, включая функциональные зависимости, преобразуется в схему с теми же зависимостями.

Трансляция — проектирование схемы, когда требования приложений выражаются средствами одной модели данных, а реализация осуществляется с помощью другой модели.

Реорганизация является частным случаем реструктуризации, когда схемы S_s и S_t изоморфны.

Конвертирование определяется так: для данной схемы S_s, соответствующей модели M_s и ассоциированной с ней БД получить схему S_t, соответствующую модели M_t и ассоциированной с ней БД.

Детализируем «операционные» понятия.

Взгляд (View) представляет собой подсхемы различных пользователей в многопользовательском режиме.

Трансформация: для данной схемы S_s модели M_s найти схему S_t, соответствующую модели M_t, которая может быть использована для операций над БД со схемой S_s. Это может быть отображение взгляда в иерархическую и сетевую модели.

Гомогенная РБД — построение глобальной схемы из локальных (однотипных) схем, покрывающих все другие схемы.

Гетерогенная РБД — общий случай интеграции локальных БД в распределенную БД (РБД).

Остальные преобразования являются частными случаями интеграции. Из них интерес представляют трансляция, трансформация и конвертирование.

Наиболее общими являются преобразования интеграции: гомогенные и гетерогенные, подробно исследуемые позднее.

При реструктуризации схема S_t — результат применения к схеме S_s операций реляционной алгебры и реляционного исчисления с отображением структур и ограничений. Решение определяется специфическими ограничениями модели. В реляционной модели, например, реструктуризация не имеет особых трудностей, поскольку спецификация структур и ограничений, равно как и управление ими, разделено.

Так, любое отношение R_t степени n в схеме S_t есть результат функции $f(R_s)$ той же степени n отношения S_s.

Сложнее с ограничениями (функциональными зависимостями). Пусть имеются в схеме S_s отношения: R(A, B), S(C, D) и ограничения $A \to B$, $C \to D$. Положим, что S_t: $T = R_{A=C} \cdot S$. Тогда в S_t имеется зависимость $A \to BD$.

Обратимость отображения не всегда имеет место,

$$S_s\text{: } R(A, B, C),\ AB \to C,\ C \to B \quad S_t\text{:}$$

$$S = \pi_{A, C}(R),\ T = \pi_{B, C}(R),\ C \to B.$$

Функциональная зависимость $AB \to C$ не включена в S_t, поскольку A, B, C не появляются в схеме одного и того же отношения. Из-за функциональной зависимости $C \to B$ соединение S и T по C образует соединение без потерь информации (S и T дает R), но возможность поддерживать ограничение $AB \to C$ утрачивается.

Отметим, что совокупность правил отображений структур и функциональных зависимостей, как показано в гл. 4, обладает полнотой и надежностью: достоверные исходные данные дают достоверные результаты.

Реструктуризация может быть проведена и для других МД.

Сложность конвертирования заключается в том, что оно оперирует не виртуальными, а реальными объектами. Это имеет место, например, при создании из двух БД одной, причем операции могут осуществляться на физическом или логическом уровне.

Трансформация реляционной МД в иерархическую и сетевую показана в гл. 5—7. Такая процедура с операциями может быть использована при переходе от концептуальной модели, которая ближе всего к реляционной МД, к иерархической и сетевой МД.

9.2. ВЫБОР МОДЕЛЕЙ ДАННЫХ

Второй проблемой является выбор модели данных и СУБД. Действительно, можно взять лишь два критерия: быстродействие и удобство обновления. По первому критерию следует предпочесть иерархическую МД, тогда как по второму — реляционную.

На самом деле критериев значительно больше: стоимость, производительность, неизбыточность и т. д.

На выбор влияет и множество факторов: 1) типы элементов данных; 2) интерфейс пользователя; 3) структура и отношения данных; способы манипулирования данными; 4) целостность БД и защита данных; 5) поддержка программная и техническая; 6) коммерческая поддержка; 7) критерии качества (надежность, точность, соответствие промышленным стандартам); 8) возможности роста и развития.

Сюда следует добавить, что не все требования могут быть сформулированы одинаково четко, а одним и тем же требованиям могут соответствовать разные МД.

В силу сложности задачи выбора СУБД ее целесообразно решать в два этапа: выбор МД; выбор СУБД в рамках принятой МД.

При решении первой задачи возникает проблема выбора по векторному критерию. Ее можно решать различными методами: методом анализа иерархий Саати Т., с помощью матриц (таблиц) принятия решений. Решение задачи последним способом приведено в работе [39]. Следует отметить, что этот вариант решения достаточно трудоемок. К тому же на результат решения сильно влияют неформальные факторы.

Названные обстоятельства привели к широкому применению упрощенного варианта решения, в котором используются следующие суждения.

Для баз данных, реализуемых на суперЭВМ, иногда называемых *мейнфреймами*, самым важным требованием, предъявляемым к БД, является быстродействие. В силу этого предпочтительным является использование иерархической модели данных или ее разновидности — многомерной модели данных. По тем же причинам для хранилищ данных предпочтительна многомерная модель.

Для операционных баз данных, реализуемых на персональных компьютерах (ПК), важнейшим требованием является простота обновления данных. В этом отношении предпочтительны реляционная, объектно-ориентированная или объектно-реляционная модели данных.

Объектно-ориентированные модели данных только начинают использовать в России. В настоящее время фактически единственной СУБД такого класса является CACHE.

Объектно-реляционные СУБД используют преимущественно гибридную разновидность.

В этих условиях задача выбора СУБД (для ПК) для построения операционных БД сводится практически к выбору среди реляционных СУБД. Этот выбор в итоге определяется требованиями, предъявляемыми к БД и формулируемыми в техническом задании на разработку базы данных.

Сравнительные характеристики некоторых реляционных СУБД приведены в табл. 9.2.

Таблица 9.2

Сравнительная характеристика некоторых реляционных СУБД

Характеристика	Access	InterBase	FoxPro	Paradox
Предельный объем, Гбайт	1	10		
Число полей	255	1000	255	255
Число индексов	32	65536	255	255
Длина поля, знаков	255	32	255	255
Длина строки, кбайт	2	64	64	4
Ссылочная целостность	Да	Нет	Да	Да
Режим клиент—сервер	Нет	Да	Нет	Нет

На окончательный выбор СУБД по-прежнему влияет много неформальных факторов. В связи с этим, по мнению авторов, целесообразно, использовать [1] такую последовательность.

1. Выбрать СУБД, подходящие по их техническим характеристикам (прежде всего — по объему данных в разрабатываемой базе данных).

2. Из получившегося набора СУБД следует отобрать:

а) по категории конечного пользователя (непрограммист; имеющий квалификацию в программировании; программист; администратор БД);

б) по развитости (удобству) интерфейса СУБД;

в) по качеству средств разработки БД (гибкость и полнота процедуры создания интерфейса пользователя и реализации алгоритма приложения, мощности языка программирования);

г) по качеству средств обеспечения целостности и защиты данных;

д) по характеристикам формирования распределенной БД и групповой работе с БД (прежде всего — режима клиент—сервер);

е) по поддержке стандартных интерфейсов связи с БД — через язык SQL и приложение ODBC;

ж) по видам блокировки данных;

к) по имиджу фирмы — разработчика СУБД.

В ряде случаев [1] используют понятие «производительность», т. е. быстродействие БД. Она предполагает оптимизацию процедуры запроса (глава 4), которая проводится далеко не во всех СУБД. В частности, такая процедура реализована в СУБД FoxPro.

Для выбора по этом параметру необходимы данные испытаний по специальным тестам [1] (TPC-A, TPC-B, TPC-C, TPC-D и TPC-E) с вычислением отношения цена/производительность. Производительность оценивается в количестве транзакций в секунду на стандартных операциях обновления (при одинаковых объемах данных). К сожалению, результаты такого тестирования публикуются редко.

К тому же повышения быстродействия можно проще достичь с использованием индексов, поэтому выбор по производительности практически не проводят.

Перейдем к реализации баз данных на физическом уровне.

9.3. ВОПРОСЫ ПРОГРАММНОЙ РЕАЛИЗАЦИИ БД, ОРГАНИЗАЦИЯ ХРАНЕНИЯ И ДОСТУП

Метод физического доступа — совокупность программных средств, обеспечивающих возможность хранения и выборки данных, расположенных на физических устройствах.

Пользователи стандартных СУБД обычно не проводят проектирование физической БД. Однако в большеразмерных и распределенных СУБД (например, Oracle) ведется распределение областей памяти. В силу этого знание характеристик физического расположения данных полезно.

Выделяют три основных режима работы приложений, связанных с использованием баз данных.

Режим 1. Получить все данные (последовательная обработка).

Режим 2. Получить уникальные данные (например, одну запись), для чего используют произвольный доступ (кэширование, идентификаторы), индексный метод (первичный ключ), произвольный доступ, последовательный доступ (бинарное B-дерево, B^+-дерево).

Режим 3. Получить некоторые данные (группу записей), для чего применяют вторичные ключи, мультисписок, инвертированный метод, двусвязное дерево.

К физической модели предъявляются два основных противоречивых требования: 1) высокая скорость доступа к данным; 2) простота обновления данных.

Еще одним относительно новым требованием является объем дополнительно используемой (вторичной) памяти.

Введем необходимые дополнительные термины.

Для ускорения процесса поиска и упорядочения данных создаются индексные файлы. В качестве индексов могут выступать отдельные поля, прежде всего — ключи. Индексный файл меньше по размеру, и потому скорость поиска увеличивается. «Платой» за это является дополнительная, вторичная память. *Индекс* может быть многоуровневым (B^+-деревья). Часто в качестве индексов используют числа.

Если одно и то же поле используется в индексе и для упорядочения записей файла, то индекс называют *основным*, а файл — индексно-последовательным. В противном случае индекс называют *вторичным*.

Если используется хотя бы один вторичный индекс, файл называют *инвертированным*.

Если вторичные индексы существуют для *всех* возможных полей, файл — *полностью инвертированный*.

В физической БД следует рассматривать методы размещения (запись и хранение) данных и методы доступа (поиск) к ним.

Основными методами хранения и поиска являются физически последовательный, прямой, индексно-последовательный и индексно-произвольный. Для их сравнительной оценки используем два критерия [12].

Эффективность хранения — величина, обратная среднему числу байт вторичной памяти, необходимого для хранения одного байта исходной памяти.

Эффективность доступа — величина, обратная среднему числу физических обращений, необходимых для осуществления логического доступа.

Выделяют [9] первичные (физически последовательный и произвольный) и вторичные (мультисписковый, инвертированный файлы, двусвязное дерево) методы доступа.

Применительно к иерархической МД основные методы получили свои названия и аббревиатуры (рис. 9.2):

Рис. 9.2. Методы хранения и доступа

1) иерархически последовательный метод доступа (Hierarchical Sequential Access Method — HSAM);

2) иерархический прямой метод (Hierarchical Direct Access Method — HDAM);

3) иерархический индексно-последовательный метод (Hierarchical Indexed Sequential Access Method — HISAM);

4) иерархический индексно-произвольный метод (Hierarchical Indexed Direct Access Method — HIDAM).

Дадим этим методам краткую характеристику [9].

Иерархически последовательный метод. Записи хранятся в логической последовательности, файл имеет постоянный размер, указатели могут отсутствовать. Данные хранятся в главном файле, а обновление требует создания нового главного [4, 10, 11] файла с упорядочением, для чего используется вспомогательный файл. Эффективность использования памяти близка к 100%, эффективность доступа низка.

Метод удобен для режима 1, однако быстродействие в режиме 2 мало: для его повышения необходимо использовать бинарный поиск (B- и B^+-деревья). Время включения и удаления записей значительно.

Разновидностью метода является физически связанный (связанно-последовательный) метод, который называют также *плотным*. Имеются несмежные физические участки данных, используются указатели. Осуществляется динамическое перераспределение памяти: в одной и той же области памяти можно выполнять несколько процессов или работать с данными, обладающими большой измен-

чивостью. При включении и удалении данных меняется только логика поиска. В режиме 1 время поиска мало, в режиме 2 — большое. Затраты на вторичную память невелики. Применять бинарный поиск здесь нельзя из-за неплотной упаковки файлов и невозможности вычислять адреса.

В организации данных используется целая группа индексных методов, основными из которых являются индексно-последовательный и индексно-произвольный методы.

Индексно-последовательный метод. Индексный файл упорядочен по первичному ключу (главному атрибуту физической записи). Индекс содержит ссылки не на каждую запись, а на группу записей (рис. 9.3). Последовательная организация индексного файла допускает в свою очередь его индексацию — многоуровневая индексация (рис. 9.4).

Процедура добавления возможна в двух видах.

1. Новая запись запоминается в отдельном файле (области), называемой *областью переполнения*. Блок этой области связывается в цепочку с блоком, к которому логически принадлежит запись. Запись вводится в основной файл.

2. Если места в блоке основного файла нет, запись делится пополам и в индексном файле создается новый блок.

Наличие индексного файла большого размера снижает эффективность доступа. В большой БД основным параметром становится

Рис. 9.3. Индексно-последовательный метод

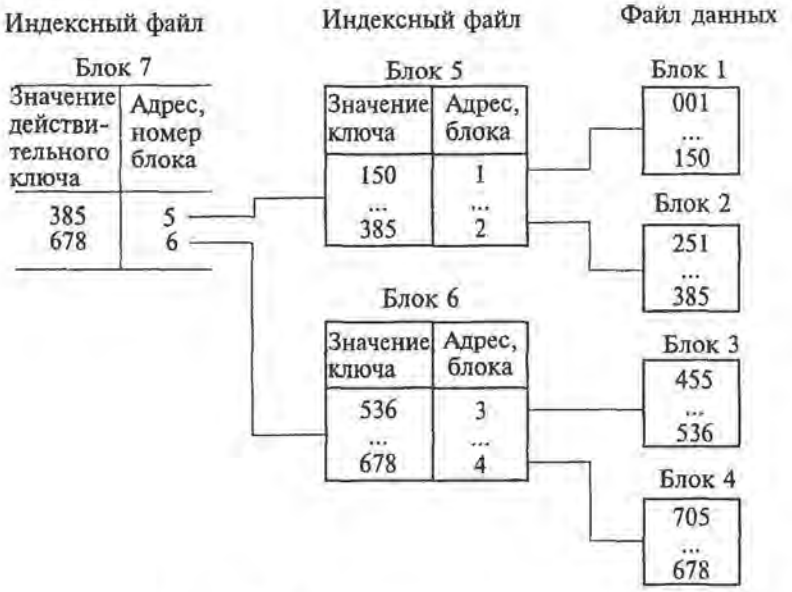


Рис. 9.4. Многоуровневая индексация

скорость выборки первичных и вторичных ключей. При большой интенсивности обновления данных следует периодически проводить реорганизацию БД. Эффективность хранения зависит от размера и изменяемости БД, а эффективность доступа — от числа уровней индексации, распределения памяти для хранения индекса, числа записей в БД, уровня переполнения.

При произвольных методах записи физически располагаются произвольно.

Индексно-произвольный метод. Записи хранятся в произвольном порядке. Создается отдельный файл, хранящий значение действительного ключа и физического адреса (индекса). Каждой записи соответствует индекс. Общая схема показана на рис. 9.5. Идея метода отражена на рис. 9.6: между вырабатываемым (относительным) адресом и физической записью (абсолютным адресом) существует взаимооднозначное соответствие.

Разновидностью этого метода является наличие *плотного* индекса: кроме главного файла создается вспомогательный файл, называемый плотным индексом. Он состоит из записи (v, p) для любого значения ключа v в главном файле, где p — указатель на запись главного файла со значением ключа v.

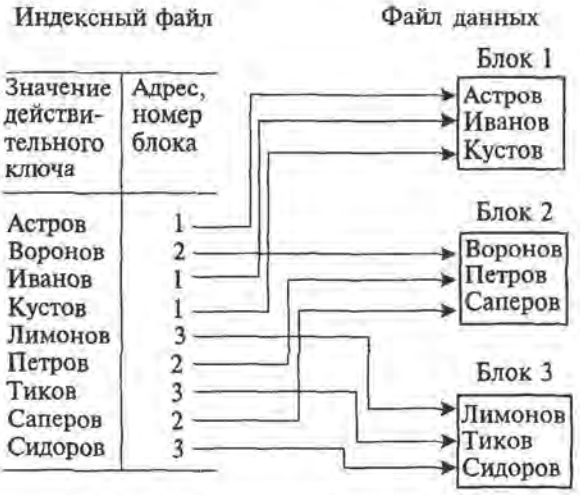


Рис. 9.5. Индексно-произвольный метод

Рис. 9.6. Идея метода кэширования:
ПР — программа рандомизации

Прямой метод. Имеется взаимооднозначное *соответствие* между ключом записи и ее физическим адресом (рис. 9.7). Выделяют два вида адресов (рис. 9.8): относительный и абсолютный.

В этом методе необходимо преобразование и надо четко знать исходные данные и организацию памяти. Ключи должны быть уникальными.

Эффективность доступа равна 1, а эффективность хранения зависит от плотности ключей.

Если не требовать взаимооднозначности, то получается разновидность метода с использованием кэширования — быстрого поиска данных, особенно при добавлении элементов непредсказуемым образом.

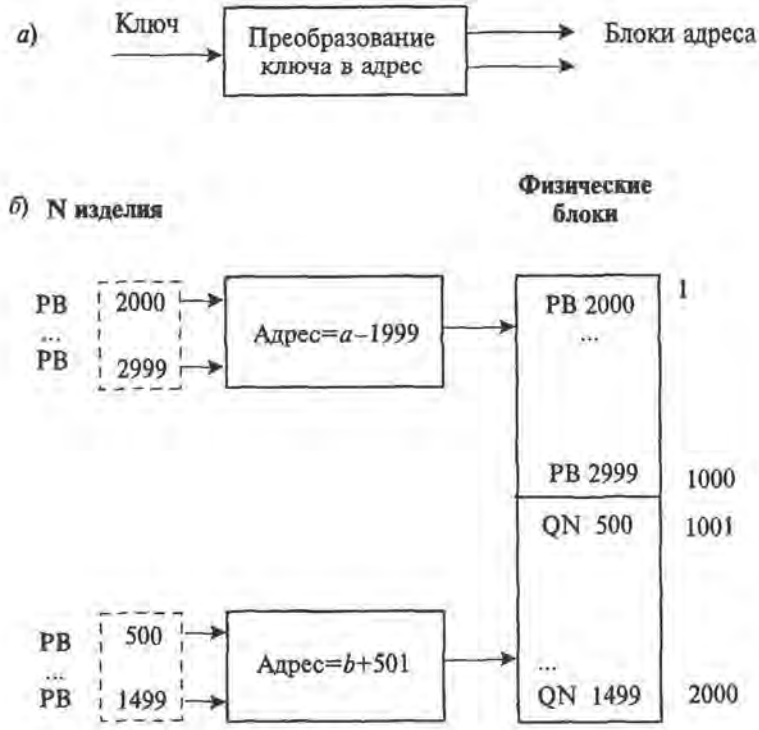


Рис. 9.7. Прямой метод

Рис. 9.8. Абсолютный и относительный адреса

Кэширование — метод доступа, обеспечивающий прямую адресацию данных путем преобразования значений ключа в относительный или абсолютный физический адрес.

Адрес является функцией значения поля и ключа записи. Записи, приобретающие один адрес, называются *синонимами*. В качестве критериев оптимизации алгоритма кэширования могут быть: потеря связи между адресом и значением поля, минимум синонимов. Память делится на страницы определенного размера, и все синонимы помещаются в пределах одной страницы или в область переполнения. Механизм поиска синонимов — цепочечный.

Любой элемент кэш-таблицы имеет особый ключ, а само занесение осуществляется с помощью кэш-функции, отображающей ключи на множество целых чисел, которые лежат внутри диапазона адресов таблицы.

Кэш-функция должна обеспечивать равномерное распределение ключей по адресам таблицы, однако двум разным ключам может соответствовать один адрес. Если адрес уже занят, возникает состояние, называемое *коллизия*, которое устраняется специальными алгоритмами: проверка идет к следующей ячейке до обнаружения своей ячейки. Элемент с ключом помещается в эту ячейку.

Для поиска используется аналогичный алгоритм: вычисляется значение кэш-функции, соответствующее ключу, проверяется элемент таблицы, находящийся по указанному адресу. Если обнаруживается пустая ячейка, то элемента нет.

Размер кэш-таблицы должен быть больше числа размещаемых элементов. Если таблица заполняется на шестьдесят процентов, то, как показывает практика, для размещения нового элемента проверяется в среднем не более двух ячеек. После удаления элемента пространство памяти, как правило, не может быть использовано повторно.

Простейшим алгоритмом кэширования может являться функция $f(k) = k \pmod{10}$, где k — ключ, целое число (табл. 9.3). Такая функция не дает равномерного распределения, и потому используют более подходящие функции, например f = сумма цифр ключа (mod 10) + 1.

«Платой» за скорость поиска и обновления в режиме 2 является нарушение упорядоченности файла, потеря возможности выполнять пакетную обработку по первичному ключу. Время обработки в режиме 1 велико.

Эффективность хранения и эффективность доступа при использовании кэширования зависит от распределения ключей, алгоритма кэширования и распределения памяти.

Таблица 9.3

Прямой метод доступа

Исходные ключи	Преобразованные объектные ключи	Адрес хранения (относительный N блока)
X101	01	01 X101
X102	02	02 X102
X103	03	03 X103
...	...	...
X199	99	99 X199
Y500	100	100 Y501
Y501	101	101 Y501

9.4.ДОСТУП К ДАННЫМ И ИХ ОБНОВЛЕНИЕ

Следует отдельно поговорить о методах поиска данных [9, 10] и выдаче результатов (в промежуточную память, на терминал) для последовательных методов обработки. Здесь используют поиск с помощью дерева (бинарное B-, B^+-деревья). Бинарное дерево является разновидностью B-дерева. B-дерево допускает более двух ветвей, исходящих из одной вершины. Любая вершина состоит из совокупности значений первичного ключа, указателей индексов и (ассоциированных) данных. Указатель индекса используется для перехода на следующий, более низкий уровень вершин (рис. 9.9). «Хранимые» в вершине данные фактически представляют собой совокупность указателей данных и служат для физической организации данных, определения положения данных, ключевое значение которых хранится в этой вершине индекса. Физическая организация ветвящейся вершины B-дерева подобна физической последовательной структуре. Алгоритм работы бинарного дерева (в вершинах) представлен на рис. 9.10.

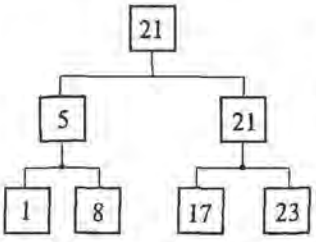


Рис. 9.9. Использование B-дерева

Этот метод дает хорошее использование памяти, обладает малым числом подопераций. Включение и удаление данных достаточно просто и эффективно.

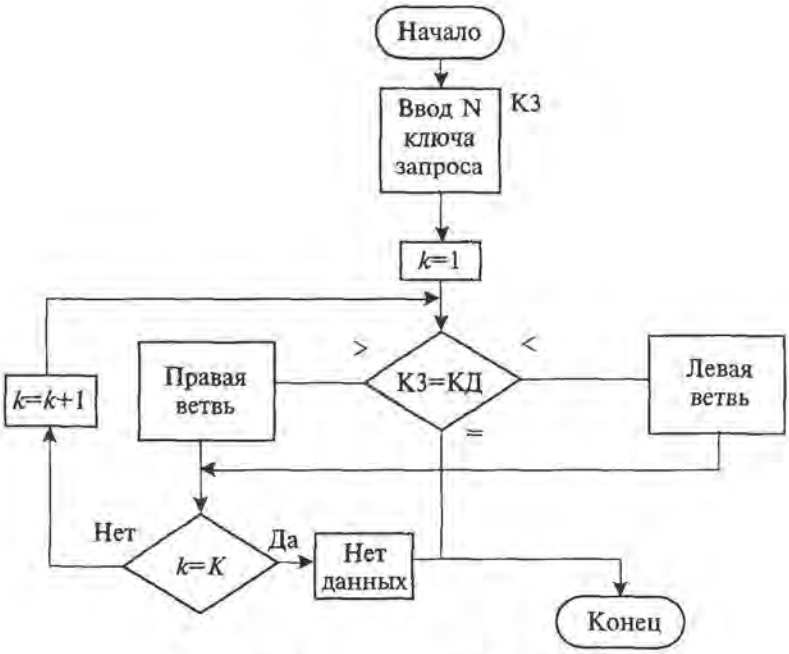


Рис. 9.10. Алгоритм поиска по B-дереву:
КЗ — ключ запроса; КД — ключ данных

Более распространенным вариантом B-дерева является B^+-дерево. Здесь возможно использовать двунаправленные указатели, а в промежуточных вершинах имеет место дублирование ключей. Если происходит деление вершины, то в исходную вершину пересылается значение среднего ключа. Фактически B^+-дерево есть индекс (указатель всех записей файла) вместе с B-деревом, как многоуровневым указателем на элементы последнего индекса.

В B^+-дереве его копия помещается в левую часть правого листа, что позволяет упростить операции добавления и удаления [8].

Подводя некоторые итоги, можно сделать следующие предварительные выводы.

Для суперЭВМ чаще всего используется иерархическая модель данных в силу ее высокого быстродействия. Для персональных компьютеров широчайшее распространение получила реляционная модель данных, по которой проведены значительные прикладные и теоретические исследования. В последнее время реляционную модель существенно потеснила объектно-ориентированная модель данных.

Контрольные вопросы

1. Назовите достоинства и недостатки иерархической, сетевой, реляционной МД.
2. Почему необходимо преобразование моделей данных? Назовите основные варианты таких преобразований.
3. Перечислите этапы выбора СУБД.
4. Какими методами возможно осуществить выбор МД?
5. Будет ли выполненный по рассмотренному методу оптимальный выбор МД оптимальным с позиции всего процесса проектирования БД?
6. Какие существуют методы организации данных и доступа к ним?
7. Дайте сравнительную характеристику последовательному, прямому, индексно-последовательному и индексно-произвольному методам.
8. Назовите первичные и вторичные методы доступа.
9. Что такое кэширование? Приведите пример алгоритма кэширования.
10. Каково назначение B- и B^+-деревьев?

Глава 10

Общая характеристика распределенных баз данных

До сих пор изучались централизованные, локальные базы данных. В то же время распределенные базы данных (РБД) находят все более широкое применение в связи с массовым распространением «сетевых» технологий.

Теория создания, использования и функционирование РБД имеет свои особенности по сравнению с централизованными БД.

В силу сказанного в данной части акцент сделан преимущественно на отличия теории РБД от теории централизованных БД.

В гл. 10 рассмотрена суть РБД и ее возможные структуры. Отмечено все более широкое распространение режима клиент—сервер, для которого представлена специфика одно- и многоуровневой структуры.

10.1. НОВЫЕ ТРЕБОВАНИЯ, ПРЕДЪЯВЛЯЕМЫЕ К БД

Базы данных явились в значительной мере следствием развития [39] автоматизированных систем управления (АСУ). Первоначально АСУ строились по централизованному принципу: данные из источников передавались в центральный вычислительный центр с суперЭВМ и там обрабатывались. В силу этого базы данных первоначально назывались банками данных.

Сразу же выяснилось, что хотя бы часть данных необходимо обрабатывать непосредственно в источниках данных. Доля децентрализованной обработки данных неуклонно росла.

К концу 80-х годов XX в. возникли новые условия работы для БД: большие объемы информации возникли во многих местах (например, розничная торговля, полиграфическое и другие производства); источником большого количества данных мог быть и центр, но к этим данным требовался быстрый доступ с периферии (географически распределенное производство, работающее по одному гра-

фику). К тому же данные могли запрашиваться и центром, и удаленными потребителями. Появилось большое количество данных, которые должны использовать в срочных запросах, чаще всего местного характера (продажа авиа- и железнодорожных билетов).

Пример 10.1. В компьютерном интегрированном полиграфическом производстве [39] необходимой является РБД (рис. 10.1), связывающая в единое целое процесс управления комплексом различных технологических процессов. Здесь осуществляется работа не с одним, а с системой приложений.

Централизованные БД, особенно построенные на классическом подходе, не могли удовлетворить новым требованиям. Возникла потребность в сетях передачи данных и формировании сети удаленных компьютеров, в том числе сети локальных БД, т. е. в распределенных базах данных (РБД).

Быстрое распространение сетей передачи данных, резкое увеличение объема внешней памяти ПК при ее удешевлении в 80-е годы XX в., развитие возможностей персональных компьютеров, которые по своим характеристикам в 90-е годы уже превосходили суперЭВМ 80-х годов, создали необходимую базу для реализации и широкого внедрения РБД.

К достоинствам РБД относятся:

- соответствие структуры РБД структуре организаций;
- гибкое взаимодействие локальных БД;

Рис. 10.1. Схема компьютерного интегрированного полиграфического производства:

УЧ — управляющая часть; ОУ — объект управления

• широкие возможности централизации узлов;

• непосредственный доступ к информации, снижение стоимости передач (за счет уплотнения и концентрации данных);

• высокие системные характеристики (малое время отклика за счет распараллеливания процессов, высокая надежность);

• модульная реализация взаимодействия, расширения аппаратных средств, возможность использования объектно-ориентированного подхода в программировании;

• возможность распределения файлов в соответствии с их активностью;

• независимые разработки локальных БД через стандартный интерфейс.

Вместе с тем РБД обладают более сложной структурой, что вызывает появление дополнительных проблем (избыточность, несогласованность данных по времени, согласование процессов обновления и запросов, использования телекоммуникационных ресурсов, учет работы дополнительно подсоединенных локальных БД, стандартизация общего интерфейса, усложнение защиты данных) согласования работы элементов.

Серьезные проблемы возникают при интеграции в рамках РБД однородных (гомогенных) локальных БД с одинаковыми, чаще всего реляционными, моделями данных.

Проблемы значительно усложняются, если локальные БД построены с использованием различных моделей данных (неоднородные, гетерогенные РБД).

Возникла необходимость в теоретической проработке процессов в РБД.

Распределенная база данных (РБД) — система логически интегрированных и территориально распределенных БД, языковых, программных, технических и организационных средств, предназначенных для создания, ведения и обработки информации.

Это означает, что информация физически хранится на разных ЭВМ, связанных сетью передачи данных. Любой узел (участок) может выполнять приложение и участвовать в работе по крайней мере одного приложения.

В распределенной базе данных используется следующая специфическая терминология.

Архитектура клиент—сервер — структура локальной сети, в которой применено распределенное управление сервером и рабочими

станциями (клиентами) для максимально эффективного использования вычислительной мощности.

Атрибут — характеристика элемента данных в объекте.

Вызов процедуры — передача управления подпрограмме или процедуре с последующим возвратом к основной программе по окончании выполнения подпрограммы или процедуры.

Вызов удаленной процедуры (Remote Procedure Call) — вызов процедуры с другого компьютера.

Инкапсуляция — объединение данных и программы (кода) в «капсуле», модуле.

Класс — объединяющая концепция набора объектов, имеющих общие характеристики (атрибуты).

Компонент — аналог класса в приложении Delphi.

Клиент — компьютер, обращающийся к совместно используемым ресурсам, которые предоставляются другим компьютером (сервером).

Локализация (размещение) — распределение данных по узлам (участкам) сети с учетом дублирования (наличия копий).

Локальная вычислительная сеть (ЛВС) — коммуникационная система, поддерживающая в пределах здания или некоторой другой территории один или несколько скоростных каналов передачи цифровой информации, предоставленных подключенным устройствам для кратковременного монопольного использования.

Метод — набор подпрограмм, оперирующих с данными.

Мост — устройство для соединения двух полностью идентичных подсетей в общую сеть.

Наследование — передача определенных свойств от класса к его производному.

Объект — комбинация элементов данных, характеризующихся атрибутами, и методов их обработки, упакованных вместе в одном модуле.

Полиморфизм — возможность переопределения процедуры в производном классе.

Прозрачность — устройство или часть программы, которая работает настолько четко и просто, что ее действия незаметны пользователю.

Свойство — аналог атрибута в приложении Delphi.

Сервер — узловая станция компьютерной сети, предназначенная в основном для хранения данных коллективного пользования и для

обработки в ней запросов, поступающих от пользователей других узлов.

Событие — сигнал запуска метода.

Среда (Computer Aided Software Engineering — CASE) — среда создания программного обеспечения, ориентированная на разработку программы от планирования и моделирования до кодирования и документирования.

Фрагмент логический — блок данных, однородных для транзакций с точки зрения доступа.

Фрагментация (расчленение) — процесс разбиения целостного объекта глобального типа на несколько частей (фрагментов).

Фрагмент хранимый — физическая реализация логического фрагмента.

Шлюз — устройство для соединения разнотипных сетей, работающих по разным протоколам.

Большинство требований, предъявляемых к РБД, аналогично требованиям к централизованным БД, но их реализация имеет свою, рассматриваемую ниже специфику. В частности, в РБД иногда полезна избыточность.

Дополнительными специфическими требованиями являются:

- ЯОД в рамках схемы должен быть один для всех локальных БД;
- доступ должен быть коллективным к любой области РБД с соответствующей защитой информации;
- подсхемы должны быть определены в месте сосредоточения алгоритмов (приложений, процессов) пользователя;
- степень централизации должна быть разумной;
- необходим сбор и обработка информации об эффективности функционирования РБД.

В дальнейшем К. Дейт сформулировал 12 правил для РБД [1—3].

1. Локальная автономность.
2. Отсутствие опоры на центральный узел.
3. Непрерывное функционирование (развитие) РБД.
4. Независимость РБД от расположения локальных БД.
5. Независимость от фрагментации данных.
6. Независимость от репликации (дублирования) данных.
7. Обработка распределенных запросов.
8. Обработка распределенных транзакций.
9. Независимость от типа оборудования.
10. Независимость от операционной системы.

11. Независимость от сетевой архитектуры.

12. Независимость от типа СУБД.

В дальнейшем рассмотрении РБД выделим:

- общие вопросы (состав, работа РБД);
- теоретические вопросы РБД, в которых по-прежнему выделим три составляющие (*создание, использование* и функционирование РБД).

10.2. СОСТАВ И РАБОТА РБД

Схема РБД может быть представлена в виде, показанном на рис. 10.2. В ней выделяют пользовательский, глобальный (концептуальный), фрагментарный (логический) и распределенный (локальный) уровни представления данных (рис. 10.3), определяющие сетевую СУБД [11].

Общий набор (система таблиц) данных, хранимых в РБД, приведен в табл. 10.1. Это глобальный уровень, который определяется при проектировании теми же методами, что и концептуальная модель централизованной БД.

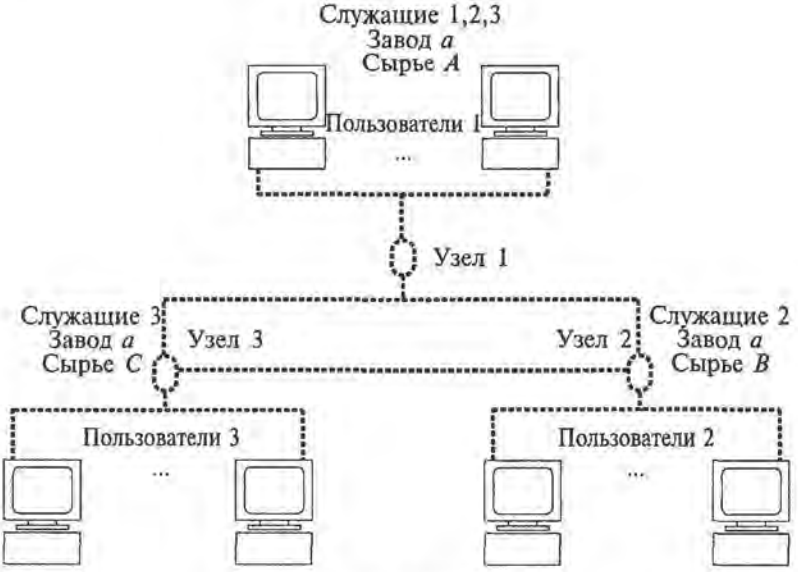


Рис. 10.2. Схема РБД

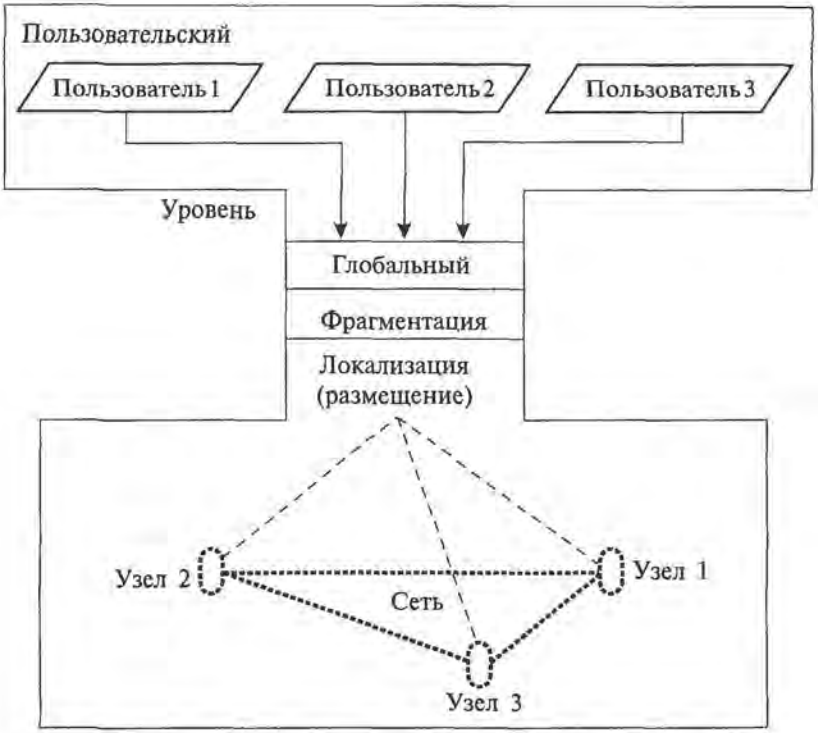


Рис. 10.3. Уровни представления данных в РБД

Таблица 10.1

Данные в РБД (глобальный уровень)

а) Служащий

N	Имя	Завод	Тариф
100	Иванов	1	6
101	Петров	1	6
102	*Сидоров*	*2*	*10*
103	*Артемов*	*2*	*12*
104	**Печкин**	**3**	**5**
105	**Крамов**	**3**	**11**

б) Завод

N	Расположение
1	С.-Петербург
2	Вологда
3	Сыктывкар

в) Сырье

N	Название	Количество
1	Картон	500
2	*Картон*	*100*
2	*Открытки*	*940*
3	**Брошюры**	**75**

В табл. 10.1 разными шрифтами выделены фрагменты БД.

Не все данные глобального уровня доступны конкретному пользователю. Наиболее полные данные (пользовательский, внешний уровень) имеются в узле 1 головного предприятия. В узлах (участках) 2, 3 данные менее полные. Так, в узле 3 они имеют вид, показанный в табл. 10.2.

Таблица 10.2

Пользовательский уровень в РБД

а) Служащий

N	Имя	Завод	Тариф
104	**Печкин**	**3**	**5**
105	**Крамов**	**3**	**11**

б) Завод

N	Расположение
1	С.-Петербург
2	Вологда
3	Сыктывкар

в) Сырье

N	Название	Количество
3	**Брошюры**	**75**

Пользовательский уровень состоит из фрагментов (например, строки 1, 2, 3 таблица «Завод» табл. 10.1) глобального уровня, которые составляют *фрагментарный*, логический уровень.

Выделяют горизонтальную и вертикальную фрагментации (расчленение). *Горизонтальная фрагментация* связана с делением данных по узлам. Горизонтальные фрагменты не перекрываются. *Вертикальная фрагментация* связана с группированием данных по задачам.

Фрагментация чаще всего не предполагает дублирования информации в узлах. В то же время при размещении фрагментов по узлам (*локализации*) распределенного уровня в узлах разрешается иметь копии той или иной части РБД. Так, например, локализация для примера в табл. 10.1 может иметь вид, показанный в табл. 10.3.

Таблица 10.3

Локализация данных

Имя таблицы	Распределение фрагментов по узлам
Служащие	1
	1, 2
	1, 3
Завод	1, 2, 3
	1, 2, 3
	1, 2, 3
Сырье	1
	2
	3

Очевидно, что для таблицы «Завод» осуществляется дублирование, а для таблицы «Сырье» — расчленение.

После размещения данных каждый узел имеет локальное, узловое представление (локальная логическая модель). Физическую реализацию (логического) фрагмента называют *хранимым фрагментом*.

Рис. 10.4. Схема работы РБД

Иначе говоря, РБД можно представить в виде, показанном на рис. 10.3.

Сеть в РБД образуют сетевые операционные системы (например, Windows NT, Novell NetWare). В качестве СУБД, изначально предназначавшихся для использования в сети, следует назвать BTrieve, Oracle, InterBase, Sybase, Informix.

В силу распределенности данных особую значимость приобретает словарь данных (справочник) РБД, который в отличие от словаря централизованной БД имеет распределенную, многоуровневую структуру.

В общем случае могут быть выделены сетевой, общий внешний, общий концептуальный, локальные внешние, локальные концептуальные и внутренние составляющие словаря РБД.

Естественно, что для работы в РБД необходимы администраторы РБД и локальных БД, рабочими инструментами которых являются перечисленные словари.

Схема работы РБД показана на рис. 10.4.

Пользовательский запрос, определяемый приложением, поступает в систему управления распределенной базы данных (СУРБД), через сетевую и локальную операционные системы попадает в локальную СУБД. Если запрос связан с локальными данными, СУБД осуществляет вызов дан-

ных из локальной БД, которые поступают пользователю. Если часть данных для выполнения приложения находится в другой локальной БД, локальная СУБД дополнительно через локальные и сетевую операционные системы осуществляет удаленный вызов процедуры (Remote Procedure Call — PRC), после выполнения которой данные передаются пользователю.

Возможны четыре стратегии хранения данных: централизованная (часто обеспечиваемый архитектурой клиент—сервер), расчленение (фрагментации), дублирование, смешанная.

Сравнительные характеристики стратегий хранения приведены в табл. 10.4. На ее основе может быть построен простейший алгоритм выбора стратегии, приведенной на рис. 10.5.

Таблица 10.4

Стратегии хранения данных

Название	Суть стратегии	Достоинство	Недостатки
Централизация (в том числе технология клиент—сервер)	Единственная копия в одном узле	Простота структуры	Скорость обработки ограничена одним узлом Ограниченный доступ Малая надежность Долговременная память определяет объем БД
Локализация (расчленение)	Единственная копия, расчленение по узлам (полная копия БД не допускается)	Объем БД определяется памятью сети Снижение стоимости РБД Время отклика при параллельной обработке уменьшается Малая чувствительность к узким местам Повышенная надежность при высокой локализации данных	Запрос может быть по всем узлам Доступ хуже, чем при централизации **Рекомендации применения:** долговременная память ограничена по сравнению с объемом БД; должна быть повышена эффективность функционирования при высокой степени локализации

Продолжение табл. 10.4

Название	Суть стратегии	Достоинство	Недостатки
Дублирование	В каждой локальной БД полная копия	Выше надежность, доступ и эффективность выборки, простота восстановления. Локальная асинхронная обработка в узлах Получение быстрых ответов	Объем БД ограничен долговременной памятью Потребность синхронизации многих копий Потребность в дополнительной памяти Слабая реализация параллельной обработки **Рекомендации применения:** фактор надежности превалирует; БД невелика; интенсивность обновления невысока; интенсивные запросы
Смешанная	Несколько копий хранимого логического фрагмента в каждом узле	Любая степень надежности Большая доступность Меньше пересылок данных Параллельная обработка	Надо хранить словари Рост стоимости согласования данных Разная частота обращения узла к различным частям БД Потеря надежности из-за расчленения данных Малая свободная долговременная память из-за дублирования

Отметим, что в обычной сети имеет место равноправие компьютеров, что может вызвать дополнительные осложнения в части доступа к данным в процедурах обновления и запросов.

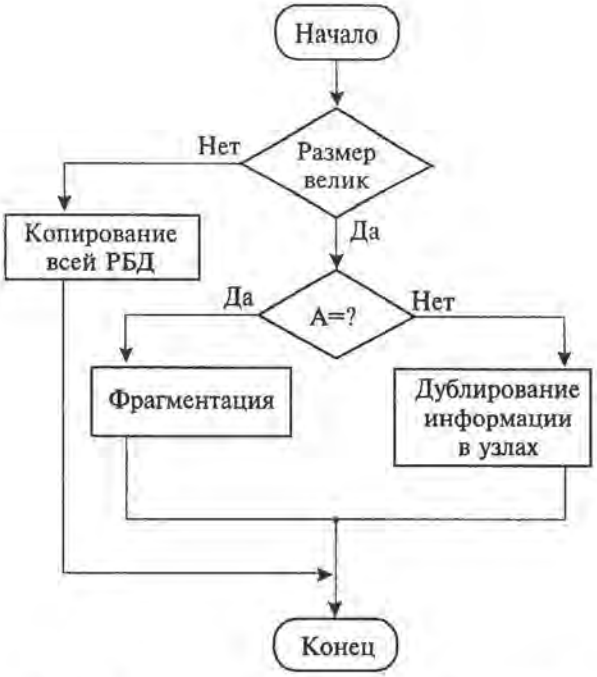


Рис. 10.5. Алгоритм выбора стратегии хранения:
А — запрос локален

В связи с этим часто используют *архитектуру клиент—сервер* (рис. 10.6) — структуру локальной сети, в которой применено распределенное управление сервером и рабочими станциями (клиентами) для максимально эффективного использования вычислительной мощности.

В этой структуре один из компьютеров, имеющий самый большой объем памяти и наиболее высокое быстродействие, становится приоритетным, называемым *сервером*. На сервере чаще всего хранятся только данные, запрашиваемые клиентами.

К *клиентам* не предъявляются столь жесткие требования по памяти и быстродействию. На них располагаются словари и приложения, служащие своеобразными фильтрами для данных сервера. В связи с этим обмен информацией в архитектуре (рис. 10.6) фактически минимизируется.

Работа в архитектуре клиент—сервер может поддерживаться и с помощью схемы Open DataBase Connectivity (ODBC), как пока-

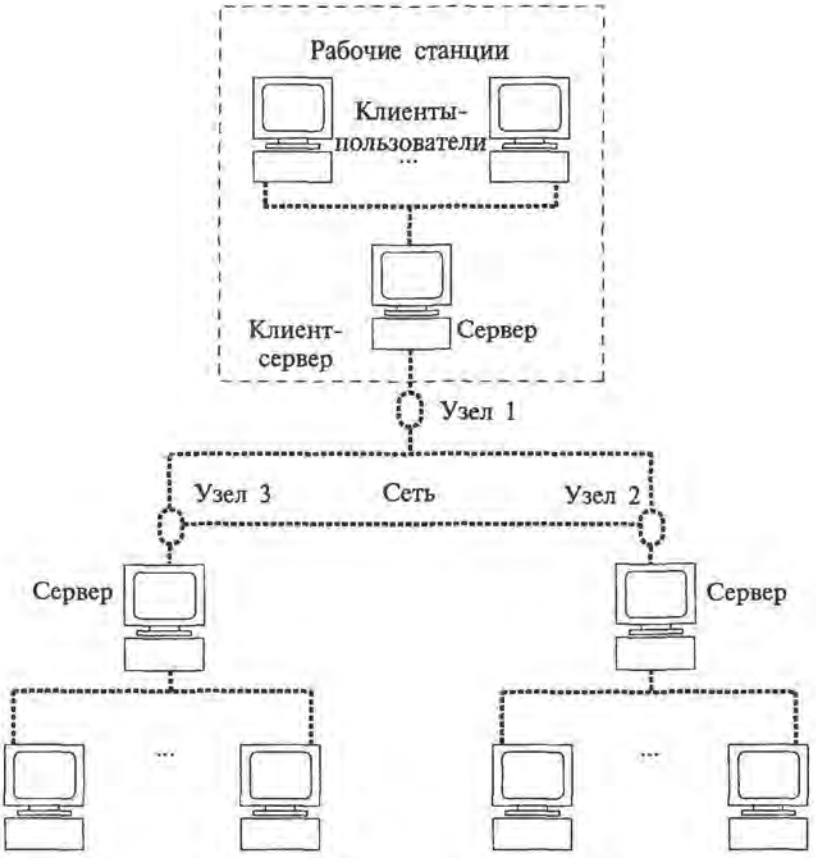


Рис. 10.6. Архитектура клиент—сервер

зано на рис. 10.7. В этом случае сеть образуется путем соединения серверов. Такое соединение обеспечивается или средствами СУБД (SQL Server) или мониторами транзакций (TUXEDO).

Обсудим более подробно вариант реализации РБД — архитектуру клиент—сервер.

Рис. 10.7. ODBC в архитектуре клиент—сервер

10.3. СИСТЕМА КЛИЕНТ—СЕРВЕР

Совместно с термином «клиент—сервер» используются три понятия.

1. Архитектура: речь идет о концепции построения варианта РБД.
2. Технология: говорят о последовательности действий в РБД.
3. Система: рассматриваются совокупность элементов и их взаимодействие.

Об архитектуре клиент—сервер говорилось ранее.

Технология клиент—сервер позволяет повысить производительность труда:

- сокращается общее время выполнения запросов за счет мощного сервера;
- уменьшается доля и увеличивается эффективность использования клиентом (для вычислений) центрального процессора;
- уменьшается объем использования клиентом памяти «своего» компьютера;
- сокращается сетевой трафик.

К таким крупномасштабным системам предъявляются следующие требования: 1) гибкость структуры; 2) надежность; 3) доступность данных; 4) легкость обслуживания системы; 5) масштабируемость приложений; 6) переносимость приложений (на разные платформы); 7) многозадачность (возможность выполнения многих приложений).

Отметим, что архитектуре клиент—сервер предшествовала архитектура файл—сервер, в которой возможны следующие варианты.

1. На компьютерах-клиентах имеется копия БД. Работа по такому варианту имеет следующие сложности: синхронизация данных различных копий в конце работы БД; высокий трафик (потоки данных между сервером и клиентами, поскольку передается в любом случае содержимое всей БД).

2. В СУБД Access, которая изначально создана как локальная, предусмотрен режим деления базы данных. Таблицы остаются на сервере (back-end), а остальные объекты (запросы, отчеты) передаются клиентам (front-end). В этом случае по-прежнему большой трафик, в силу чего при использовании файл-серверов количество подключаемых клиентов — при их надежной работе — до четырех.

В то же время требовалось подключение десятков и даже сотен клиентов [1—3]. Этого удалось достичь в архитектуре (режиме, технологии) клиент—сервер. В этом режиме трафик резко уменьшается, поскольку по сети передаются только те данные, которые соответствуют запросам клиентов [2].

Для этого пришлось построить СУБД, изначально предназначенные для работы в сети. Фирма Microsoft [27—29] вынуждена была — в дополнение к СУБД Access, которая использовалась с помощью приложения ODBC только для клиентских целей — предложить в качестве сервера Microsoft SQL Server. Такая структура оказалась тяжеловесной и неудобной, так как разработчику требовалось знать уже две СУБД.

Из других предложений очень удачным оказался программный продукт Delphi [30—36], в рамках которого могут использоваться СУБД dBase, Paradox, и, при отдельной инсталляции, InterBase (режим клиент—сервер). При этом СУБД InterBase поддерживается, наряду с языком программирования SQL, мощным, понятным, простым и широко распространенным языком программирования (Object) Pascal [11], построенным с применением объектно-ориентированного подхода [33].

Последнее обстоятельство позволяет легко строить объектно-реляционные базы данных (см. гл. 9). Высокая степень автоматизации программирования дает возможность резко упростить и снизить трудоемкость процедур создания интерфейса пользователя, и особенно алгоритма приложения.

В системе клиент—сервер возможно выделить следующие составляющие: сервер, клиент, интерфейс между клиентом и сервером, администратор.

Сервер осуществляет управление общим для множества клиентов ресурсом. Он выполняет следующие задачи:

- управляет общей БД;
- осуществляет доступ и защиту данных, их восстановление;
- обеспечивает целостность данных.

К БД на сервере предъявляются те же требования, как и к централизованной многопользовательской БД.

Следует отметить, что результаты запросов клиента помещаются в рабочую область памяти сервера, которая в ряде СУБД (например, Oracle) называют «табличная область». Поскольку она не занимает много места, для каждого клиента-пользователя целесообразно создавать свою табличную область. В этом случае исходные таблицы становятся для пользователя недоступными, а архивация (копирование) БД приложения клиента упрощается.

Клиент хранит в компьютере свои приложения, с помощью которых осуществляется запрос данных на сервере. Клиент решает следующие задачи:

- предоставляет интерфейс пользователю;
- управляет логикой работы приложений;

• проверяет допустимость данных;

• осуществляет запрос и получение данных с сервера.

Средством передачи данных между клиентом и сервером является **сеть** (коаксиальный кабель, витая пара) с сетевым (сетевая операционная система — СОС) и коммуникационным программным обеспечением.

В качестве СОС могут использоваться Windows NT, Novell NetWare. Коммуникационное программное обеспечение позволяет компьютерам взаимодействовать на языке специальных программ — коммуникационных протоколов.

В общем случае такое взаимодействие осуществляется с помощью семиуровневой схемы ISO с соответствующими протоколами. Для локальных сетей схема упрощается. Протоколоми для Windows NT служит Transmission Control Program/Internet Program (TCP/IP), для NetWare — Sequenced Packed eXchange/Internet Packed eXchaned (SPX/IPX).

Разнообразие сетевых средств делает необходимым создание стандартного промежуточного программного обеспечения клиент—сервер, находящегося на сервере и клиентах. Говорят о прикладном программном интерфейсе (Application Programming Interface — API). Сюда относятся Open DataBase Connectivity (ODBC) и Integrated Database Application Programming Interface (IDAPI), используемый в приложении Delphi и СУБД InterBase.

Взаимодействие клиентов и сервера можно представить себе следующим образом.

При обращении пользователя к приложению компьютер-клиент запрашивает у пользователя имя и пароль. После этого — при правильном ответе — приложение может быть запущено клиентом. Приложение дает возможность подключиться к серверу, которому сообщается имя и пароль пользователя.

Если подключение осуществлено, начинает работать сервер, выполняющий два вида процессов: переднего раздела и фоновые.

Процессы переднего раздела непосредственно обрабатывают запросы, фоновая составляющая связана с управлением процессом обработки.

Работа сервера может иметь такой порядок.

1. После поступления запроса диспетчер ставит его в очередь по схеме «первым пришел — первым обслужен».

2. Процесс переднего раздела выбирает «самый старый» запрос и начинает его обработку. После завершения результаты помещаются в очередь для передачи клиенту.

3. Диспетчер посылает результаты из очереди соответствующему клиенту.

При обработке запроса фоновые процессы выполняют другие важные операции, основными из которых являются следующие:

- запись данных из БД в промежуточную (буферную) память рабочей области (при чтении) и обратно (при обновлении);
- *запись в журнал транзакций;*
- архивация (копирование) групп транзакций;
- аварийное завершение транзакций;
- периодическая запись на диск контрольных точек для обеспечения восстановления данных в РБД после аппаратного сбоя.

Администратор РБД (АРБД) должен решать следующие задачи [39].

1. Планирование РБД и распределение памяти.
2. Настройка конфигурации сети.
3. Создание РБД.
4. Работа с разработчиками *приложений.*
5. Создание новых пользователей и управление полномочиями.
6. Регулярная архивация БД и выполнение операций по ее восстановлению.
7. Управление доступом к БД с помощью ОС и СОС, средств защиты и доступа.

В больших системах АРБД может состоять из ряда лиц, отвечающих, например, за ОС, сеть, архивацию, защиту.

Таким образом, система клиент—сервер своеобразна: с одной стороны, ее можно считать разновидностью централизованной многопользовательской БД, с другой стороны, она является частным случаем РБД.

В связи с этим имеется специфика и в процессе проектирования. Оно по-прежнему начинается с создания приложения, затем — интерфейса и БД. Однако в силу специфики системы этапы фрагментации и размещения отсутствуют и есть свои особенности.

Основное ограничение для работы такой системы — минимальный трафик. Поэтому при разработке приложения, помимо обычных задач (уяснения цели приложения, логики обработки, вида интерфейса) особое внимание следует обратить на разработку DLL-сценария и распределение функций между клиентами и сервером.

Использование для составления сценария CASE-средств значительно сокращает трудоемкость работ по проектированию. Иначе эта процедура выполняется вручную с помощью команд языка SQL.

Важнейшей является задача распределения функций. По самой сути технологии на сервере расположена БД, а на компьютерах-клиентах — приложения. Однако при прямолинейных процедурах обеспечения целостности и запросах в сети может возникнуть объемный сетевой трафик.

Чтобы его снизить, возможно использовать следующие рекомендации.

1. Обеспечение целостности для всех приложений лучше централизовать и осуществлять на сервере. Это позволит не только сократить трафик, но и рационально использовать СУРБД, улучшив управление целостностью (ссылочной, ограничений, триггеров) данных.

2. Целесообразно использовать на сервере хранимые процедуры, совокупность которых можно инкапсулировать в виде пакета (модуля). В результате трафик уменьшится: клиент будет передавать только вызов процедуры и ее параметры, а сервер — результаты выполнения процедуры.

3. В ряде случаев клиентам следует получать уведомления базы данных (например, заведующему складом — о нижнем уровне запасов, при котором следует выполнять новый заказ). Если уведомление производится по запросу клиента, трафик увеличивается. Проще эту (хранимую) процедуру разместить на сервере, который будет автоматически уведомлять клиента о возникновении события. В то же время клиент при необходимости может получать информацию с помощью простых вызовов процедур.

В режиме клиент—сервер выделяют две разновидности структуры: одноуровневую, о которой шла речь до сих пор, и многоуровневую (рис. 10.8).

При использовании режима «толстого» клиента (одноуровневая структура) клиентские приложения находятся непосредственно на машинах пользователей, либо частично на сервере в виде системы хранимых процедур. Сложность клиентской части порой требует ее

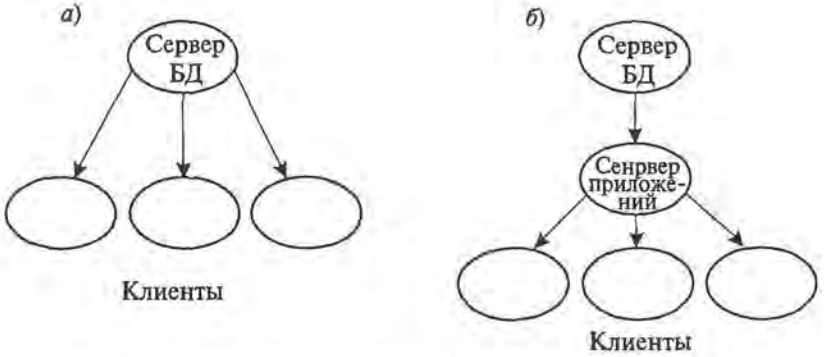


Рис. 10.8. Одноуровневая — «толстый» клиент (*а*) и многоуровневая — «тонкий» клиент (*б*) структуры клиент—сервер

администрирования. Использование системы хранимых процедур в значительной мере снижает нагрузку на сеть. При использовании «тонкого» клиента (многоуровневая структура) приложения пользователей лежат и выполняются на отдельном мощном сервере (сервере приложений, который может быть и на одном компьютере с сервером БД), что в значительной мере снижает требования к пользовательским машинам.

Так, в СУБД InterBase в одноуровневой структуре в каждом клиенте имеется утилита Borland Database Engine (BDE — ядро Delphi) объемом 8 Мбайт. В многоуровневой структуре BDE-утилита имеется только в сервере приложений, а объем памяти, занимаемой клиентом, снижается до 212 кбайт. Достигается этот результат за счет усложнения структуры.

Контрольные вопросы

1. Каковы новые требования к БД?
2. Что такое «распределенная база данных — РБД»?
3. Что такое локальный и удаленный доступ?
4. Каковы сетевые уровни представления данных?
5. Что такое фрагментация (расчленение) данных? В чем цель горизонтальной и вертикальной фрагментации?
6. Что такое локализация (размещение) данных?
7. Назовите сетевые операционные системы.
8. Назовите марки СУБД, изначально предназначенные для работы в сети.
9. Что такое архитектура «клиент—сервер»?
10. Перечислите стратегии хранения, их достоинства и недостатки, рекомендации по выбору стратегии.
11. Что такое однородные и неоднородные РБД? Особенности интеграции локальных БД в РБД.
12. За счет чего повышается производительность труда в системе клиент—сервер?
13. В чем состоят задачи, решаемые сервером? клиентом?
14. Назовите операционные системы и коммуникационное программное обеспечение системы клиент—сервер.
15. Как взаимодействуют клиенты и сервер?
16. Каковы задачи администратора системы?
17. Назовите разновидности структуры режима клиент—сервер и дайте их сравнительную характеристику.

Глава 11

Создание РБД

Специфические этапы создания РБД — это фрагментация и локализация. Проектные решения на этих этапах неоднозначны, и в ряде случаев (при выборе структуры РБД) полезно использовать математическое моделирование.

РБД состоит из связанных локальных баз данных (ЛБД). Возможны два варианта создания РБД: проектирование РБД «с нуля»; объединение (интеграция) уже готовых ЛБД. В первом варианте модели данных ЛБД, как правило, одинаковы (однородные ЛБД), во втором модели данных могут быть различны (неоднородные ЛБД).

Рассмотрена теория интеграции для однородных и неоднородных ЛБД как совокупности процессов описания и манипулирования данными. Представлены правила преобразования моделей данных друг в друга.

11.1. ОБЕСПЕЧЕНИЕ ЦЕЛОСТНОСТИ

Будем придерживаться порядка изложения в соответствии с этапами проектирования (см. рис. 2.6).

В качестве принципов проектирования можно использовать:

- максимум локализации данных и сокращение количества пересылаемых по кратчайшему пути данных: рекомендуется иметь до 90% ее в локальной БД (ЛБД) узла и около 10% — в ЛБД других узлов;
- локальность расположения данных следует определять по отношению к наибольшему числу приложений.

В качестве критериев проектирования РБД могут быть [16]: 1) минимум объема пересылаемых данных и сообщений; 2) минимум стоимости трафика; 3) минимум общего времени, необходимого для обслуживания запросов к БД.

В рассмотрении РБД возможно выделить два случая работы: с одним приложением и с системой приложений. Возможно нисходящее и восходящее проектирование.

Восходящее проектирование используется обычно в случае, когда РБД создается из уже работающих локальных БД. Эти особенности освещены в проблеме интеграции однородных и неоднородных БД.

Восходящее проектирование может быть этапом нисходящего проектирования, которое, в силу того что оно применяется гораздо чаще, обсудим подробнее.

В общем случае целостность данных может нарушаться по следующим основным причинам:

- ошибки в создании структуры локальных БД и их заполнении;
- просчеты в построении структуры РБД (процедуры фрагментации и локализации);
- системные ошибки в программном обеспечении взаимодействия локальных БД (одновременный доступ);
- аварийная ситуация (неисправность технических средств) и восстановление РБД.

Первая позиция подробно освещена ранее и здесь рассматриваться не будет. Специфика остальных трех позиций для РБД может быть зафиксирована в виде совокупности проблем (рис. 11.1). Чет-

Рис. 11.1. Проблемы РБД

вертая позиция исследуется в гл.12, вторая и третья — подробно рассмотрены здесь.

Иногда к нарушению целостности относят умышленное искажение информации, т. е. несанкционированный доступ. Эти вопросы будут изучены в гл. 12.

11.2. ФРАГМЕНТАЦИЯ И ЛОКАЛИЗАЦИЯ

Напомним (см. рис. 2.6), что общая этапность проектирования РБД напоминает этапность при создании централизованной БД и отличие имеет место лишь в этапах фрагментации (расчленения) и локализации (размещения).

Основными факторами, определяющими методику расчленения, являются допустимый размер каждого раздела; модели и частоты использования приложений; структурная совместимость; факторы производительности БД. Связь между разделом БД и приложениями характеризуется идентификатором типа приложения, идентификатором узла сети, частотой использования приложения и его моделью.

Сложность реализации этапа размещения БД определяется многовариантностью. Поэтому на практике рекомендуется в первую очередь рассмотреть возможность использования определенных допущений, упрощающих функции СУРБД (например, допустимость временного рассогласования БД, осуществление процедуры обновления БД из одного узла).

Фрагментация [2], как отмечалось ранее, может быть горизонтальной и вертикальной. Фрагмент может быть определен последовательностью операции селекции и проекции реляционной алгебры (см. гл. 5). При декомпозиции следует выполнить ряд условий.

1. **Полнота** — все данные глобального отношения R должны быть отображены в его фрагменты.

2. **Восстанавливаемость** — всегда возможно восстановить глобальное отношение из фрагментов.

3. **Непересечение** — целесообразно, чтобы фрагменты не пересекались (дублирование производится на этапе локализации).

При горизонтальной фрагментации с помощью селекции любое подмножество кортежей объединено общностью свойств, определяемых описанием предметной области.

Вертикальная фрагментация с помощью проекции делит глобальное отношение (схему R) по приложениям (или по географическому признаку).

Фрагментация корректна, если любой атрибут глобального отношения (схемы R) присутствует в каком-либо подмножестве атри-

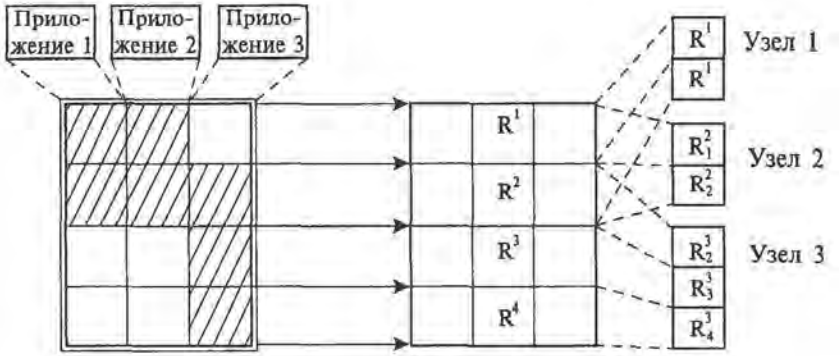


Рис. 11.2. Схема фрагментации и локализации данных

бутов и глобальное отношение восстанавливается естественным соединением.

Фрагментация совместно с локализацией (рис.11.2) определяет в конечном итоге быстроту реакции РБД на запрос.

Обозначим узлы через $j\ (j = \overline{1, J})$, а приложения — через $k\ (k = \overline{1, K})$. Если имеется отношение r со схемой R, то в узле j имеется отношение-фрагмент R^j. Если в узле j имеется к тому же копия фрагмента R_i, $i\ (i = \overline{1, I})$, то обозначим ее через R_i^j. Тогда схема фрагментации и локализации может быть представлена в виде, показанном на рис. 11.2.

Иными словами, при фрагментации декомпозиция глобального отношения должна обладать свойством соединения без потерь.

Возможна смешанная фрагментация, которой соответствует совокупность операций селекции и проекции

$$R_i^{jk} = \sigma_{A = F}(\pi_{A, B, C}(R)),$$

где $F = \{a_{1j}, ..., a_{mj}\}$; m — число записей в горизонтальном фрагменте.

После фрагментации осуществляют локализацию. В качестве критерия локализации [2] удобно использовать влияние локализации (размещения) на задачу оптимизации запроса при известной его структуре. Для этого необходимы моделирование и оптимизация всех приложений для любого варианта размещения.

Воспользуемся обозначениями, введенными в данной главе. Пусть f_{kj} — частота активизации приложения k в узле j; r_{ki} — число ссылок поиска приложения k во фрагменте i; u_{ki} — число ссылок обновления данных приложения k во фрагменте i; $n_{ki} = r_{ki} + u_{ki}$.

Задача размещения имеет две основные разновидности: без использования и с использованием копий.

Рассмотрим первую разновидность. Здесь возможны эвристические и строго математические алгоритмы.

Обсудим сначала один из эвристических алгоритмов размещения, состоящий из нескольких шагов.

Шаг 1. Используем метод «наиболее подходящего» размещения: фрагмент R_i размещаем в узле j, где число ссылок на него максимально. Число локальных ссылок

$$B_{ij} = \sum_{k=1}^{K} f_{kj} n_{ki} \to \max. \qquad (11.1)$$

Из выражения (11.1) определяем узел j*, где следует разместить фрагмент.

Шаг 2. Применим метод выделения «всех выгодных узлов» для избыточного размещения: помещаем R_i во всех узлах, где стоимость ссылок приложений, осуществляющих поиск, больше стоимости ссылок приложений, обновляющих данные во фрагменте R_i в любом узле: $B_{ij} > 0$ или

$$\sum_{k=1}^{K} f_{kj} r_{ki} > C \sum_{k=1}^{K} \sum_{\substack{j'=1, \\ i \neq j}}^{J} f_{kj'} u_{ki}, \qquad (11.2)$$

где C — отношение стоимости обновления и поиска.

Шаг 3. Используем метод «добавочного копирования». Пусть d_i — *степень избыточности* R_i; F_i — выгодность размещения копии в любом узле РБД и $\beta(d_i) = (1 - 2^{1-d})F_i$. Модифицируя выражение (11.2), получим

$$B_{ij} = \sum_{k=1}^{K} f_{kj} r_{ki} - C \sum_{k=1}^{K} \sum_{\substack{j'=1, \\ i \neq j}}^{J} f_{kj'} u_{ki} + \beta(d_i) > 0. \qquad (11.3)$$

Учтем вертикальную фрагментацию.

Пусть схема R_i декомпозирована на R_s и R_t, где s и t — узлы.

Используем следующие рассуждения.

1. Если существует два приложения $П_s$ и $П_t$, которые используют только атрибуты R_s и R_t, т.е. обращаются только к узлам s и t, то результатом фрагментации и локализации будет отсутствие удаленных ссылок.

2. Если имеется приложение (множество приложений) Π_{q1}, локальное для узла w, которое ссылается на R_s или R_t, то появится одна удаленная ссылка.

3. Если имеется приложение Π_{q2}, локальное по отношению к w и ссылающееся на атрибуты R_s и R_t, то получатся две удаленные ссылки.

4. Если имеется приложение Π_{q3} в узлах, отличных от w, s или t, и ссылающееся на R_s и R_t, то появится еще одна удаленная ссылка.

В общем случае выгодность фрагментации и локализации (при C = 1)

$$B_{ist} = \sum_{k\in\Pi_s} f_{ks} n_{ki} + \sum_{k\in\Pi_t} f_{kt} n_{ki} - \sum_{k\in\Pi_{q1}} f_{kw} n_{ki} - \sum_{k\in\Pi_{q2}} f_{kw} n_{ki} - \sum_{\substack{k\in\Pi_{q3},\\ i\neq w,s,t}} f_{ki} n_{ki} > 0. \tag{11.4}$$

Описанные эвристические алгоритмы могут не дать рационального решения, поэтому рассмотрим некоторые строгие алгоритмы, базирующиеся на целочисленном программировании [16].

Введем обозначения: $i\ (i = \overline{1, I})$ — независимые файлы-данные; $j\ (j = \overline{1, J})$ — узлы; L_i — объем файла; b_j — объем памяти узла (для файлов); d_{sj} — коэффициенты, учитывающие расстояние между узлами $s\ (s = \overline{1, J})$ и j ($d_{ss} = 0$); r_{sj} — стоимость передачи; λ_{ij} — интенсивность запросов к файлу i из узла j; λ'_{ij} — интенсивность корректировки сообщений; α_{ij} — объем запросов к файлу i из узла j; β_{ij} — объем запрашиваемых данных при выполнении запроса i из узла j;

$$x_{ij} = \begin{cases} 1, \text{если файл i находится в узле j,} \\ 0, \text{в остальных случаях.} \end{cases}$$

Тогда объем данных, поступающих в узел j, содержащий файл i, при выполнении запроса к этому файлу с учетом интенсивности равен $\lambda_{ij}(\alpha_{ij} + \beta_{ij})(1 - x_{ij})$, а объем данных, составляющих запросы и ответы, $\lambda_{ij}(\alpha_{ij} + \beta_{ij})(1 - x_{ij})x_{ij}$.

Возможны следующие критерии: объем передаваемых данных; общая стоимость трафика.

При использовании первого критерия получаем следующую задачу целочисленного программирования:

$$F = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{s=1}^{J} \lambda_{ij}\left(\alpha_{ij} + \beta_{ij}\right) d_{sj}\left(1 - x_{ij}\right) x_{ij} \to \min;$$

$$\sum_{j=1}^{J} x_{ij} = 1;$$

$$\sum_{i=1}^{I} L_i x_{ij} \le b_j.$$

Аналогичные выражения получаются при использовании второго критерия.

«Суммируя» представленные задачи, можно решить задачу размещения и определить количество копий.

Более детальные результаты получены с помощью теории массового обслуживания и целочисленного программирования (см. [21—23]).

11.3. ПРОЦЕСС ИНТЕГРАЦИИ

РБД могут быть [10] однородными (рис. 11.3, *а*) и неоднородными (рис. 11.3, *б*).

Неоднородность определяется следующими обстоятельствами:

1) использование разных моделей данных;

2) отличие в синтаксисе и семантике даже одного вида моделей (например, реляционных);

3) различие методов управления БД (резервирования и восстановления, синхронизации, блокировки).

Независимо от однородности в общем случае данные из (любой) ЛБД с одной моделью данных могут быть переданы в (любую) ЛБД с другой моделью данных.

В связи с этим следует обсудить общую постановку задачи взаимного преобразования данных.

Первоначально для построения неоднородных БД использовались процедуры выгрузки—загрузки фактически на физическом уровне. Информация:

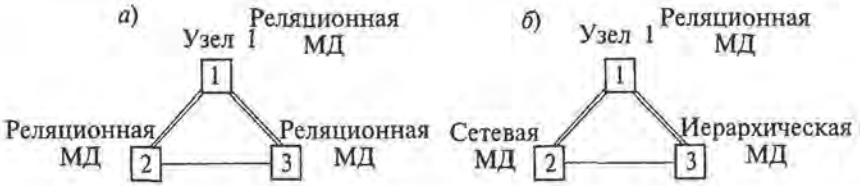


Рис. 11.3. Интеграция однородных (*а*) и неоднородных (*б*) локальных БД

• выгружалась из исходной аппаратно-программной среды;

• запоминалась в общем для исходной и объектной сред формате (например, ASCII);

• загружалась в объектную среду.

На этом пути встретились серьезные трудности. Если преобразование файлов с последовательной физической организацией достаточно просто, хуже обстоит дело при использовании произвольной выборки. Применение индексно-последовательных файлов практически исключает возможность автоматизации процедуры преобразования.

В этом случае необходимо сначала перейти к произвольному или последовательному способу доступа с уничтожением всех указателей и вспомогательных программных средств, осуществить преобразование, а затем — ввести новую систему указателей.

Реализация оказалась сложной, к тому же терялось свойство *физической независимости*.

Дальнейшие работы пошли по пути преобразований на логическом уровне, при этом выявились два направления.

1. Построение схем «попарного» взаимодействия моделей данных. При наличии n локальных БД требуется $n(n-1)$ таких схем, что чрезвычайно неудобно.

2. Использование универсальной виртуальной промежуточной модели. Данные преобразуются в эту глобальную модель, и обратно. В качестве такой модели сначала использовали ER-диаграммы.

В дальнейшем более перспективным и успешным оказался современный вариант применения (глобальной) реляционной модели [14, 15], о котором поговорим детальнее.

Для формального описания этого варианта возможно использовать аппарат коммутативной алгебры (рис. 11.4).

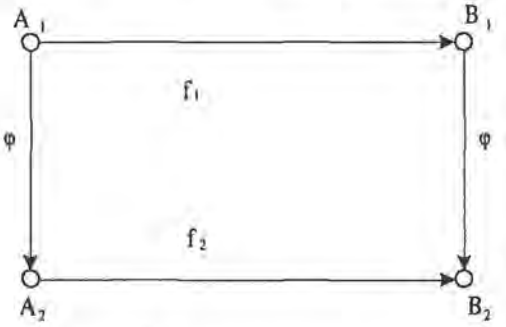


Рис. 11.4. Коммутативная диаграмма

Пусть A_1 и B_1 — состояния системы (например, в прямоугольных координатах), а f_1 — функция преобразования состояния системы. Пусть A_2, B_2 и f_2 описывают ту же систему (например, в полярных координатах). Пусть φ — правило преобразования одной системы координат в другую. Для этого случая в строгих алгебраических исследованиях показано, что преобразование верно, если диаграмма на рис. 11.4 коммутативна, т. е. выполняется условие

$$f_2 \times \varphi = \varphi \times f_1, \qquad (11.5)$$

где $\times$ — некоторая алгебраическая операция.

Можем полагать, что A_1 и A_2 — исходные таблицы СУБД с разными, в общем случае, моделями данных, к которым осуществляется запрос; B_1 и B_2 — результаты операций обновления или запроса; f_1 и f_2 — правила получения данных (ЯОД или ЯМД или язык запроса); φ — правила перехода (по данным) из одной БД в другую.

Для «точечных», физических систем математические выкладки коммутативной алгебры выполнены детально.

Использование условия (11.5) для БД значительно труднее. Дело в том, что физические системы оперируют однородными, бесструктурными множествами состояний A_i и B_i, $i = 1, 2$.

В РБД осложнения вызываются следующими обстоятельствами [14, 15].

1. Множества A_1 и A_2 структурированы, при этом структура определяется принятой моделью данных. Даже при использовании различных СУБД с одной и той же моделью данных (например, реляционной) возможны различия в форматах данных, в командах ЯОД.

2. ЯМД и языки запросов могут быть различны даже для однородных РБД с разными типами ЛБД.

Часть этих трудностей уже преодолена. Так, проблема различия в ЯОД и ЯМД при использовании реляционных БД различного типа фактически снята при использовании «стандартного» языка SQL2 [26] или приложения Open DataBase Connectivity (ODBC). Сложнее обстоит дело с другими моделями данных, еще труднее, если в РБД используются различные модели данных ЛБД.

Перечисленные проблемы интеграции ЛБД [14, 15] обсудим в самом общем виде.

11.4. ПРЕОБРАЗОВАНИЕ СТРУКТУРЫ И ДАННЫХ

В общем случае преобразование данных модели M_i в данные модели M_j возможно двумя путями:

1) «попарное» преобразование, обычно с помощью драйверов: здесь при наличии n локальных БД требуется n(n − 1) драйверов; этот путь широко используется в приложении ODBC;

2) построение универсальной промежуточной виртуальной модели $M_в$ с преобразованием по схеме

$$M_i \to M_в \to M_j,$$

при этом в качестве $M_в$ может выступать ER-диаграмма или реляционная модель [14, 15].

Для анализа сути преобразования данных рассмотрим процессы построения БД и работы с ней.

1. Первоначально на этапе концептуального проектирования задается некоторая схема S_i проектируемой БД и на выбранном ЯОД создается структура БД. Эта структура определяется множеством V_i БД, где элементы $v_i \in V_i$ есть типы данных; Id — множество идентификаторов (имен) полей. Тогда состояние $B_i = (Id \to V_i)$.

2. Общая схема S_i БД определяется набором Ms_i состояний

$$Ms_i: S_i \to B_i.$$

Исходное состояние B_i обозначим E_i.

3. Каждому значению V_i ставится в соответствие значение поля данных, что будет подразумеваться при дальнейших рассуждениях.

4. При работе с БД используется выбранный ЯМД. Оператор o_i этого языка осуществляет преобразование $E_i \to B_i$ или $B_i \to B_i'$. Множество операторов o_i — через Mo_i. Тогда

$$Mo_i: B_i \to B_i' \text{ или } E_i \to B_i,$$

где E_i — начальное состояние. Тогда модель данных (МД)

$$M_i = < S_i, Ms_i, o_i, Mo_i >.$$

Ее необходимо преобразовать в модель M_j, обладающую отмеченными ранее характеристиками.

Преобразование назовем *правильным*, если справедливы свойства:

1) полная определенность, т. е. любое состояние модели M_i представимо в модели M_j;

2) интерпретируемость: любой оператор ЯМД в M_i имеет интерпретацию в ЯМД M_j;

3) воспроизводимость: любое изменение БД в M_i выполняется некоторым оператором изменения, воспроизводимого средствами M_j.

Рассмотрим эти свойства детальнее.

1. Введем операторы $\sigma: S_i \to S_j$; $\psi: B_i \to B_j$. Тогда должна быть коммутативна диаграмма на рис. 11.5, *а*, *б* или

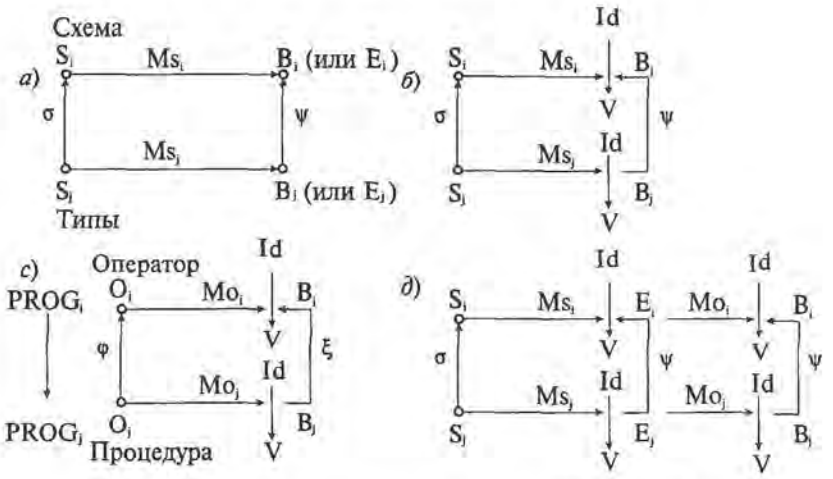


Рис. 11.5. Коммутативные диаграммы ЯОД и ЯМД

$$Ms_i \cdot \sigma = \psi \cdot Ms_j.$$

Отметим, что B_i и B_j в конечном счете характеризуются множествами $V_i(S_i) = \{v_i^1, ..., v_i^k\}$ и $V_j(S_j) = \{v_j^1, ..., v_j^m\}$. В простейшем случае $k = m$ и говорят о *подобных* типах данных. Однако он встречается редко, о чем свидетельствует [14] пример СУБД Access и «подобных» реляционных СУБД (табл. 11.1—11.4). Здесь лучше говорить не о первичных типах данных, а о некоторых классах (табл. 11.3, 11.4) производных типов данных и построении отображения этих классов.

Таблица 11.1

Типы данных СУБД Access

Тип данных	Характеристика
Текстовый	255 байт
Memo	64 кбайт
Числовой	1, 2, 4, 8 байт
Дата/Время	8 байт
Денежный	8 байт
Счетчик	4 байта
Логический	1 байт
Объект OLE	До 1 Гбайт

Таблица 11.2

Соотношение типов данных СУБД Paradox и Access

Paradox	Access
Alphanumeric	Текстовый
Number	Числовой с плавающей запятой, 8 байт
Short Number	Числовой; целое
Currency	Денежный
Date	Дата/Время

Таблица 11.3

Соотношение типов данных СУБД BTrieve и Access

BTrieve	Access
String, lstring, zstring	Текстовый
Integer 1 byte 2 byte 4 byte	 Числовой. Байт Числовой; целое Числовой; длинное целое
Float, bifloat 4 byte 8 byte	 Числовой с плавающей точкой Числовой с плавающей точкой
Decimal, numeric	Числовой с плавающей точкой
Money	Денежный
Logical	Да/Нет
Lvar	Объект OLE

Иными словами, речь идет о непересекающихся множествах вида $C_i(V_i) = \{C_i^r\}$, $C_i^r \in V_i$, $r = \overline{1, R}$ и $C_j(V_j) = \{C_j^s\}$, $s = \overline{1, S}$, при этом $C_i^r \to C_j^s$ без потери информации. Тогда имеет место посредством оператора ψ отображение $C_i(V_i(S_i)) \to C_j(V_j(S_j))$, где $S_j = \sigma(S_i)$, при этом отображение σ (преобразование типов) должно быть *биективно*.

2. Обеспечение интерпретируемости предполагает, что реализация программы $PROG_i$ (или оператора o_i) на ЯМД M_i эквивалентна

Таблица 11.4

Соотношение типов данных языка SQL и СУБД Access

SQL	Access
Char	Текстовый
Varchar	Текстовый
Tinint	Числовой. 4 байта
Smallint	Числовой; целое
Integer	Числовой; длинное целое
Real	Числовой с плавающей точкой
Float	Числовой с плавающей точкой
Double	Числовой с плавающей точкой
Date	Дата/Время
Time	Дата/Время
Timestamp	Да/Нет
Image	Объект OLE

реализации программы $PROG_j$ на ЯМД M_j, т. е. коммутативна диаграмма на рис. 11.5, *с*.

Тогда общая связь ЯОД и ЯМД различных моделей данных может быть представлена в виде схемы, показанной на рис. 11.5, *d*, где p_j — процедура.

3. Говорят, что функция φ *сохраняет операторы*, если для любого $o_i \in Mo_i$ процедура $p_j = \varphi(o_i)$ такова, что

а) исходные состояния $b_i \in B_i$ и $b_j \in B_j$ связаны зависимостью $b_i = \psi(b_j)$;

б) оператор o_i переводит состояние b_i в состояние b_i', а оператор p_j — *состояние* b_j в b_j' при этом $b_i' = \psi(b_j')$.

Оператор o_i и композиция операторов p_j, удовлетворяющая данному требованию, называются *эквивалентными* относительно отображения ψ. Именно такое отображение ψ нас будет интересовать.

Функция ξ — верификационная: если исходные состояния e_i и e_j, когда начинают функционировать программы, связаны соотношением $e_i = \psi(e_j)$, то действия, произведенные программой $prog_i \in PROG_i$, переводят систему M_i в состояние b_i, а действия

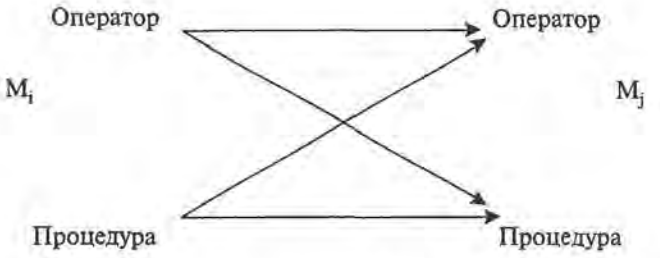


Рис. 11.6. Соотношение операторов и процедур моделей данных

$\tau.prog_i \in PROG_j$ переводят БД M_j в состояние b_j и $b_i = \psi(b_j)$, τ — функция отображения программ.

Возможные соотношения операторов и процедур показаны на рис. 11.6.

Назовем набор операторов o_i в ЯМД M_i *функционально полным* (ФП), если для любого начального состояния $b_i' \in B_i$ произвольной БД со схемой S_i можно задать последовательность операций $prog_i$, переводящую БД в произвольное заданное состояние $b_i \in B_i$, удовлетворяющее схеме S_i.

Между этим понятием и свойством воспроизводимости действий существует связь. Если есть ФП-набор операторов M_i в M_j, то $M_j \to M_i$ обладает свойством полной определенности и является воспроизводимым.

Далее предполагаем, что свойства 1—3 и функциональная полнота имеют место.

11.5. ОДНОРОДНЫЕ И НЕОДНОРОДНЫЕ РБД

Рассмотрим случай однородных локальных БД ($\sigma \equiv 1$). В этом случае чаще всего говорят о реляционных БД. Однако даже при выполнении всех сделанных ранее предположений имеются затруднения.

Пусть имеет место строгое соответствие $Ms_i \cdot \sigma = \psi \cdot Ms_j$. При переходе к произвольной модели M_k (рис. 11.7) из-за $\psi \neq 1$ должно быть $Ms_i \cdot \sigma = \psi \cdot Ms_k$. Если сохранить ψ (что желательно), то семантика ЯОД M_i должна быть очень богатой.

Это условие выполнить трудно, поэтому действуют иначе. Изменяют функцию ψ для ЯОД и для ЯМД путем построения в ODBC соответствующих (попарных) драйверов. С позиций языков полезно использование стандартного языка SQL, многочисленные диалекты которого хорошо «понимают» друг друга. Поскольку разно-

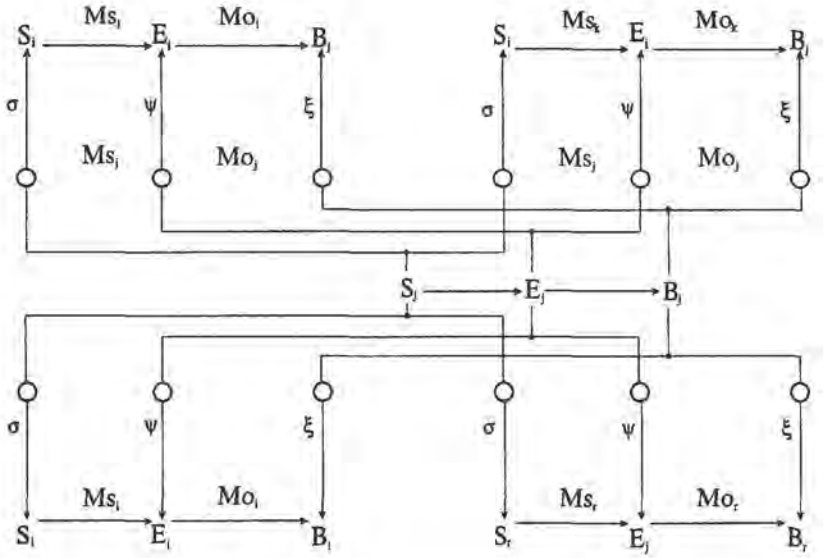


Рис. 11.7. Соотношение локальных баз данных

видностей однотипных СУБД немного (для реляционных БД это обычно Paradox, dBASE, FoxPro, BTrieve, Access, Oracle), то немного и используемых в ODBC драйверов, что служит серьезным аргументом в пользу данного направления.

Рассматриваемая процедура существенно усложняется для случая разных моделей данных ($\sigma \neq 1$).

Построение неоднородных РБД возможно двумя способами.

1. Пользователю в каждом узле предоставляется интерфейс от пользователей других локальных БД, т. е. $n(n-1)$ схем преобразования данных.

2. Создается единый стандартный пользовательский интерфейс и стандартная внутренняя форма запроса: одна схема РБД, но каждому типу локальных БД должен соответствовать свой тип преобразователя схемы в общую форму (n типов преобразования для n локальных СУБД). Пользователь должен в этом случае изучить новую схему — сетевой стандарт. Оба способа имеют свои достоинства и недостатки.

В общем случае возникает необходимость в построении промежуточной, универсальной, виртуальной модели данных [14, 15], для реализации которой можно использовать, например, реляционную модель.

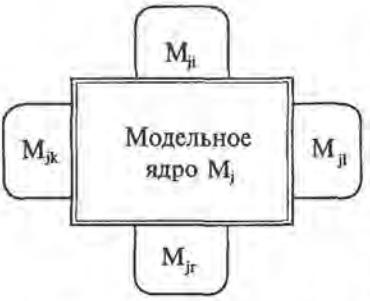


Рис. 11.8. Преобразование данных различных моделей

Сложность здесь не только, как отмечалось ранее, в необходимости богатства семантики ЯОД и ЯМД. Действительно, коммутативные диаграммы операторов ЯМД предполагают сохранение инвариантности в M_i логических зависимостей данных, имеющих место в M_j. Напрямую этому требованию семантика исходных операторов не удовлетворяет: разным внутренним моделям M_i будут соответствовать различные требования к семантике модели M_j, в том числе и для состава функционального полного набора операций. Отобразить произвольную модель M_i в M_j без *изменения семантики* данных и операций над ними в M_i не представляется возможным.

Тогда в общем случае и следует строить (рис. 11.8) промежуточную модель M_j, такую чтобы M_j = ядроM_j + M_{ji}, где модель M_{ji} отображает логические зависимости, присутствующие в M_i. Назовем M_{ji} *интерпретацией* M_i в M_j: она имеет все свойства моделей данных.

При конструировании M_{ji} полезно использовать следующие принципы.

1. Ориентация на начальные типы данных M_j при построении в M_{ji} более сложных типов, отражающих зависимости в M_i, при этом *основой* M_{ji} является M_j-*ядро*.

2. Совмещение синтаксиса множества операторов O_{ji} ЯМД M_{ji} с синтаксисом множества операторов O_j путем изменения семантики.

3. Ядро M_j должно обладать максимальным разнообразием первичных типов данных с минимумом ограничений.

4. Принципы построения ядра должны обеспечить его простое расширение введением аксиоматических типов данных и конструированием декларативно заданных семантических ограничений на первичные типы данных.

В этом случае процедура преобразования может иметь такую последовательность:

1) определяются начальная M_i и целевая (конечная) M_j модели данных;

2) отображается ЯОД M_i в M_j при инъективных операторах ψ: $B_i \rightarrow B_j$ и проверяется коммутативность диаграмм описания схем;

3) расширяется модель M_j до M_{ji}, включая выбор схем аксиом σ_{ji}, определения семантики множества операторов O_{ji}, расширения M_{ji} модели M_j и формальной проверки логической и операторной полноты;

4) строятся отображения ЯОД M_i в ЯОД M_{ji}, ЯМД M_i в ЯМД M_{ji}, включая проверку коммутативности диаграмм отображения схем и операторов.

Истинность системы аксиом должна быть инвариантна к изменениям, осуществляемым оператором ЯМД в M_{ji}. Аксиома $a_{ji} \in M_{ji}$ является инвариантом модели данных M_j по отношению к операторам O_{ji}, если из истинности $a_{ji}(R)$, где R — реализация составного типа данных, на котором определяется a_{ji}, следует истинность $a_{ji}(o_{ji}(R)/R)$ для любого $o_{ji} \in O_{ji}$ и произвольной R. Такая система аксиом должна быть логически и операторно полна.

Перечисленным требованиям расширяемой модели удовлетворяют ER-диаграммы [39] и реляционные модели [14, 15]. В последнем случае язык логики предикатов может служить основой построения аксиоматических типов данных.

Как показал опыт работ [39] и особенно [14, 15], универсальная модель достаточно объемна и сложна, а требование универсальности (в частности, для включаемых моделей бинарных отношений, семантических сетей, фреймов) излишне жестко и в практике необходимо редко.

Чаще используют (наряду с реляционными МД) только сетевые и иерархические модели данных. Их реализаций немного, и потому представляется более резонным строить соответствующие драйверы — «попарный» метод (см. рис. 11.3, *б*). Такой подход является к тому же элементом построения универсальной модели.

В связи с этим рассмотрим взаимопреобразования иерархическая — реляционная и сетевая — реляционная МД (соотношение сетевой и иерархической моделей данных достаточно просто и к тому же обсуждалось в гл. 6).

При описании реляционной БД используем стандартный вариант языка SQL, полагая, что все нестандартности могут быть устранены путем использования приложения ODBC.

При преобразовании МД предполагается (см. рис. 11.5, *а*), что отображение множеств типов данных (форматов данных) проведено. Речь пойдет об отображении структур данных и (рис. 11.5, *б*) соответствующих команд (ЯМД).

Возможны различные сочетания МД. Обсудим лишь преобразование сетевых моделей данных в реляционные, а из многочисленных вариантов преобразования сетевой МД в реляционную рассмотрим только два.

А. Вся сетевая МД считается совокупностью элементов-пар записей владелец — член [10], а структура записей предполагается простой (без агрегатов), без обратных связей.

Б. В структуре записей разрешаются агрегаты, структура наборов иерархическая и не имеет циклов [14, 15]. Фактически идет преобразование системы наборов в систему таблиц (файлов).

Напомним важнейшие характеристики (табл. 11.5), определяющие структуру сетевой МД.

Таблица 11.5

Команды сетевой МД

Размещение	Пояснение	Отключение	Включение
CALC <ключ>	Кэширование	MANDATORY — обязательное уничтожение	AUTOMATIC — автоматическое
VIA <имя> SET	Хранение записей рядом		
DIRECT	Работа программиста		
SYSTEM	По умолчанию, сингулярные записи	OPTIONAL — необязательное уничтожение, передача	MANUAL — ручное, от SYSTEM
INDEX	Используется редко	SYSTEM	

♦ **А.** Рассмотрим отдельно [10] две процедуры преобразования: сетевая — реляционная МД, реляционная — сетевая МД. В [10] такое преобразование (без операций) названо трансляцией.

Обозначим через N и R записи и отношения в сетевой и реляционной БД соответственно. Проделаем следующие действия.

1. Для любого типа записи N_i определим такое отношение:

а) R_i содержит один атрибут для любого элемента данных N_i;

б) если N_i имеет ключ, то ключ R_i соответствует ключу N_i, иначе — ключ R_i соответствует ключу БД N_i, который появляется в R_i как явный атрибут.

2. Для любой связи L_{ij}, где N_i — запись-владелец, N_j — запись-член, определим реляционное ограничение и изменим существующее отношение так, чтобы:

а) ключ N_i появился как внешний ключ в R_j;

б) ввести ограничения в зависимости от типа членства.

Пусть X — внешний ключ (foreign key) $FK(R_i)$ отношения R_i в отношении R_j. Это ограничение обозначим FC_{ij}. Ключ X может быть

и не определен (запись сетевой МД передается системе). Это факт обозначим NFC_{ij}.

Результат преобразования сетевой модели в реляционную показан в табл. 11.6 для четырех случаев: 1) MANDATORY AUTOMATIC (MA); 2) MANDATORY MANUAL (MM); 3) OPTIONAL AUTOMATIC (OA); 4) OPTIONAL MANUAL (OM).

Из табл. 11.6 видно, что преобразование возможно не для всех случаев.

Таблица 11.6

Преобразование сетевой МД в реляционную

MANDATORY AUTOMATIC	MANDATORY MANUAL	OPTIONAL AUTOMATIC	OPTIONAL MANUAL
Добавить FC_{ij}. 1. Кортеж отношения R_j включается только при существовании кортежа отношения R_i со значениями ключа, соответствующего значению $FK(R_i)$ в добавляемом кортеже R_j	Добавить NFC_{ij}. 1. Кортеж отношения R_j включается только при существовании кортежа отношения R_i со значениями ключа, соответствующего значению $FK(R_i)$ в добавляемом кортеже R_j	Добавить NFC_{ij}. 1. Кортеж отношения R_j включается только при существовании кортежа отношения R_i со значениями ключа, соответствующего значению $FK(R_i)$ в добавляемом кортеже R_j; значение $FK(R_i)$ может быть не определено	
2. При удалении кортежа R_i необходимо удалить все кортежи R_j, где $FK(R_i)$ равно значению в удаляемом отношении R_i	2. При удалении кортежа R_i необходимо удалить все кортежи R_j, где $FK(R_i)$ равно значению в удаляемом отношении R_i	2. При удалении кортежа R_i необходимо удалить все кортежи R_j, где $FK(R_i)$ равно значению в удаляемом отношении R_i, а значение $FK(R_i)$ должно быть заменено на «не определено»	2. При удалении кортежа R_i необходимо удалить все кортежи R_j, где $FK(R_i)$ равно значению в удаляемом отношении R_i, а значение $FK(R_i)$ должно быть заменено на «не определено»
3. При обновлении кортежа R_j значение $FK(R_i)$ не может заменяться на значение, которое не соответствует ключу какого-либо кортежа R_i	3. При обновлении кортежа R_j значение $FK(R_i)$ не может заменяться на значение, которое не соответствует ключу какого-либо кортежа R_i		

Преобразование реляционной модели в сетевую модель ведется по следующим правилам.

Необходимо, чтобы все атрибуты реляционной МД имели уникальные имена. В реляционной модели следует выявить внешние ключи, используемые для функциональных зависимостей, отображаемые связями между записями в сетевой схеме, и типы связей M : N. Предполагается, что связям соответствуют отдельные отношения, первичные ключи которых составлены из внешних ключей, участвующих в связях M : N.

Пусть имеются реляционная схема $\rho = \{R_1, ..., R_n\}$, первичные ключи любого отношения и считается, что функциональные типы связей представлены внешними ключами, а тип M : N отдельным отношением связи.

Тогда процедура перехода может иметь такой вид.

1. С помощью первичных ключей выявить все отношения «связи» и получить множество отношений α.

2. Для любого отношения R_i в $\rho - \alpha$:

а) сформировать тип связи N_i с ключом, соответствующим первичному ключу R_i, при этом имя отношения становится именем типа записи, атрибуты отношения — элементами данных в типе записи;

б) для любого подмножества атрибутов (исключая первичный ключ в R_i), если это подмножество представляет собой в R_i первичный ключ, принадлежащий $\rho - \alpha$ (подмножество является внешним ключом), добавить связь L_{ij} между типами записи N_i, соответствующей R_i, и N_j, соответствующей R_j, где внешний ключ есть первичный ключ.

При этом N_i становится типом записи-члена, а N_j — записи-владельца. Удалить атрибуты, соответствующие ключу R_j из N_j.

3. Присвоить связи L_{ij} уникальное имя и установить для связи тип членства:

а) сформировать тип записи N_i аналогично пункту 2а;

б) добавить по одной связи между типом записи N_i, соответствующим R_i, и любому из n других типов записей, участвовавших в связи, что определяется внешним ключом в R_i; N_i — общий тип записи-члена, а записи n — записи-владельцы в добавляемых связях (обычно n = 2).

Из табл. 11.6 следует, что преобразование возможно не для всех режимов сетевой базы данных.

♦ **Б.** В этом случае имеет место взаимооднозначность преобразования, поэтому рассмотрим только отображение сетевой МД в реляционную.

При нелинейной структуре таблицы (см. гл. 6) каждая «таблица в таблице» может рассматриваться в реляционной БД как самостоятельная подчиненная таблица. Для таких пар «вхождения» (главная — подчиненная таблицы) возможно использовать приемы, описанные ранее для линейной структуры таблиц в сетевой модели данных.

При многократном «вхождении» таблиц в работах [14, 15] предлагается использовать понятие «селекторный путь («вхождения»)». В этом случае преобразование может иметь несколько вариантов.

Преобразование реляционной модели данных в объектно-реляционную не представляет самостоятельного интереса.

Для гибридной разновидности ОРБД используется реляционная модель данных. Преобразование в расширенную реляционную модель равносильно переходу к объектно-ориентированной модели данных.

Преобразование реляционной МД в объектно-ориентированную МД характеризуется многовариантностью, поскольку одну и ту же задачу можно решить с использованием однородных или неоднородных (см. гл. 9) структур данных. Даже при использовании только неоднородных структур возможны разные варианты (ROW, ROW TYPE, их комбинации).

Преобразование объектно-ориентированной модели данных в реляционную похоже на переход от сетевой МД (при нелинейной структуре таблиц) к реляционной модели данных. Здесь фактически по-прежнему удобно использовать понятие «селекторный путь».

Контрольные вопросы

1. Как осуществляется локализация? по каким критериям? как определить количество необходимых копий в узлах?
2. Что такое интеграция в РБД? однородная? неоднородная?
3. Какой математический аппарат можно использовать для анализа интеграции?
4. В чем отличие математического описания физической системы и системы локальных БД?
5. Какие вы знаете программные средства для обеспечения однородной интеграции?
6. Как обеспечивается неоднородная интеграция?

Глава 12

Использование и функционирование РБД

Особенности использования РБД связаны со спецификой запросов. Функционирование РБД характеризуется особенностями обеспечения одновременного доступа, защиты, восстановления данных, определяемых используемой СУБД.

Перейдем к детальному изучению специфики названных процессов.

12.1. ЗАПРОСЫ

Если структура запросов заранее известна (стандартна), эффективность процесса запроса определяется на описанных ранее этапах фрагментации и локализации [11].

Если же структура запросов неизвестна, возникает необходимость оптимизации процесса запросов. Это особенно актуально при наличии параллельной обработки данных.

Целями (целевыми функциями) оптимизации могут быть:

- минимум времени передачи данных;
- минимум времени обработки данных в узлах;
- увеличение параллельности при обработке и передачах данных;
- улучшение трафика и загрузки процессоров в сети;
- минимум стоимости (задержки) только передачи данных (если имеются низкоскоростные каналы связи);
- минимум стоимости локальной обработки (если скорость передачи данных соизмерима с быстродействием процессора);
- выравнивание нагрузки компьютеров.

Для характеристики затрат на запросы введем два параметра: стоимость обработки и задержку данных.

Стоимость ТС обработки данных объема x, пересылаемых в узел i из узлов j, связанной с эксплуатацией физических каналов связи, составляет

$$TC(x) = \sum_{j=1}^{J} C_0^{ij} + C_1^{ij} x,$$

где C_0^{ij}, C_1^{ij} — определяемые системой матрицы стоимости установления связи и передачи единицы информации ($C_0^{ii} = C_1^{ii} = 0$).

Задержка TD(x) — время между началом и окончанием обработки запроса:

$$TD(x) = \sum_{j=1}^{J} D_0^{ij} + D_1^{ij} x,$$

где D_0^{ij}, D_1^{ij} — матрицы времени установления связи и передачи единицы информации.

Примем такие предположения:

1) каналы связей однородны ($C_0^{ij} = C_0$, $C_1^{ij} = C_1$, $D_0^{ij} = D_0$, $D_1^{ij} = D_1$);

2) стоимость передачи высока, и затратами на локальную обработку можно пренебречь (учитываются параметры только передачи данных).

Оптимизацию запроса можно разделить на два этапа:

1) глобальную оптимизацию, которую далее будем называть просто оптимизацией;

2) локальную оптимизацию, проводимую методами, описанными в гл. 4 и здесь не рассматриваемыми.

Для описания сути оптимизации используется аппарат реляционной алгебры (РА) с ее эквивалентными преобразованиями, позволяющими в результате сократить время ответа на запрос. Переменными оптимизации могут служить выбор физических копий и назначение узлов при выполнении операций РА.

Снова используем дерево запросов, в котором каждому из листов соответствует отношение, каждой вершине — операция РА. Конечной вершине соответствует узел РБД (локальная БД), а ответу на вопрос — корневая вершина.

Операции реляционной алгебры (селекция S или σ, проекция P или π, соединение J, объединение UN, разность DF, декартово произведение CP) и связывающие их законы описаны в гл. 4 данной работы. Полусоединение SJ рассмотрено в этой главе.

Возможно строгое решение задачи оптимизации, чаще всего сводящееся к задаче целочисленного программирования. Однако при ее решении встречаются серьезные затруднения (большой объем данных задачи и потребность значительного свободного ресурса, дефицит времени решения) и выигрыш по сравнению с эвристическими алгоритмами, как правило, невелик. К тому же на решение

влияют такие трудно учитываемые факторы, как топология сети, протоколы передачи данных, модели и схема размещения данных.

Поэтому воспользуемся эвристическим подходом к оптимизации.

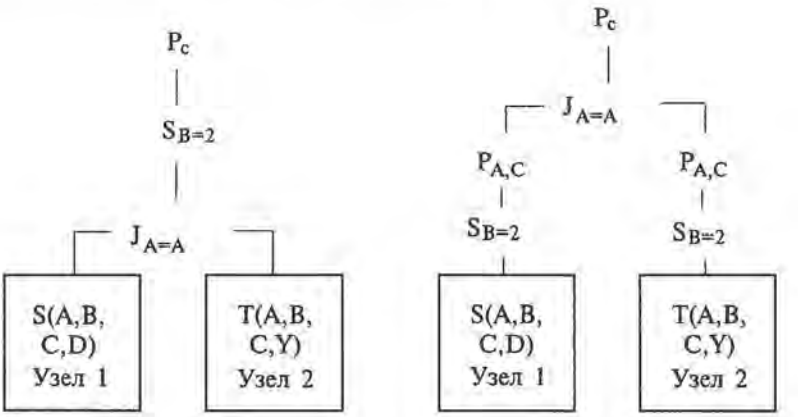
Стоимость и время пересылки данных можно сокращать за счет снижения объема передаваемой информации. В этой связи возможны следующие рекомендации.

1. Унарные операции необходимо выполнить как можно ранее (в узлах), как это показано на рис. 12.1, где S и J — унарные операции.

2. Многократно встречающиеся выражения лучше выполнить один раз. Для этого в дереве запроса проводится слияние листьев-отношений, затем — слияние одинаковых промежуточных операций (рис. 12.2). Здесь используется очевидное выражение

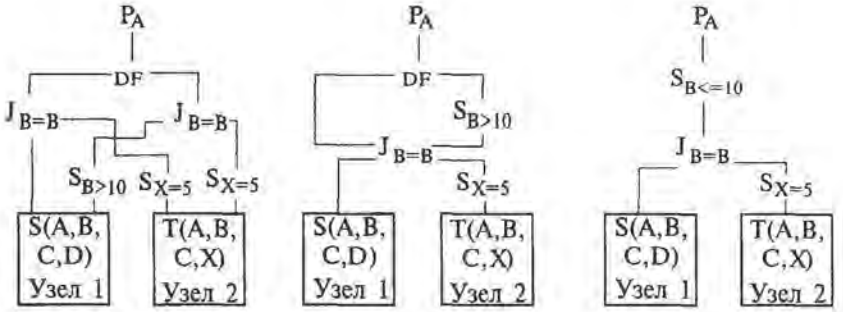
$$DF(T, S_{NF}(T)) = S_F(T), NF = NOT F.$$

3. Использовать *полусоединения*. Идея полусоединения показана на рис. 12.3. При традиционной схеме либо R надо пересылать в узел 2, либо S — в узел 1. Можно пересылать лишь один раз S, сокращенную в помощью проекции, в узел 1 и затем осуществлять полусоединение. Покажем сказанное на примере.



$$P_c(S_{B=2}(J_{A=A}(S,T)))\text{-->}$$
$$\text{-->}P_c(J_{A=A}(S_{B=2}(S),P_{A,C}(S_{B=2}(T))))$$

Рис. 12.1. Использование унарного оператора в конце (*а*) и в начале (*б*) преобразования



$$P_A(DF(J_{B=B}(S,S_{X=5}(T))),J_{B=B}(S_{B>10}(S),S_{X=5}(T))) \text{-->}$$
$$\text{-->}S_{B<=10}(J_{B=B}(S,(S_{X=5}(T))))$$

Рис. 12.2. Выделение общих подвыражений:
а — исходный запрос; *б* — промежуточный запрос; *в* — преобразованный запрос

Пусть даны следующие отношения:

Узел 1 Узел 2

r	(A	B)
	1	4
	1	5
	1	6
	2	4
	2	6
	3	7
	3	8
	3	9

s	(B	C	D)
	4	13	16
	4	14	16
	7	13	17
	10	14	16
	10	15	17
	11	15	16
	11	15	16
	12	15	16

r'	(B)
	4
	5
	6
	4
	6
	7
	8
	9

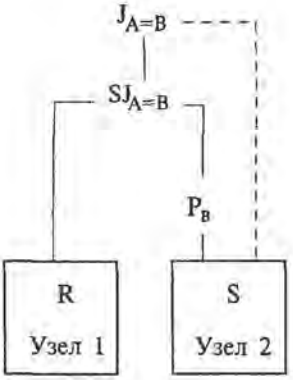
s'	(B	C	D)
	4	13	16
	4	14	16
	7	13	17

Имеются следующие возможности:

а) переслать S из узла 2 в узел 1 (24 значения);

б) вычислить $r' = P_B(r)$ в узле 1 и переслать его в узел 2, где вычислить $s' = J(r', s)$ и отправить в узел 1. Там найти J(r, s) как J(r, s'), т. е. получить полусоединение. Передается уже 9 + 6 =15 значений.

Напомним, что полусоединение r и s или SJ(r, s) есть $P_R(J(r, s))$, $R \in r$. Если отношения r и s находятся в разных узлах, то вычисление



$J_{A=B}(S,R)=J_{A=B}(S,\ SJ_{A=B}(R,P_{A=B}(S)))$

Рис. 12.3. Использование полусоединения

полусоединения уменьшает объем передаваемых данных. Заметим, что для полусоединения справедливо J(SJ(r, s), s).

Иногда полусоединение может полностью заменить соединение.

Отметим, что полусоединение чаще применяется при вертикальной фрагментации.

4. Стратегия *использования копий* (доступа). Она осуществляется на физическом уровне. Определить лучший вариант при их значительном количестве возможно только с помощью моделирования. Пусть учитываются только затраты передачи данных и имеются матрицы стоимости или времени передач. Значения матрицы могут быть определены точно, а размеры отношений — приближенно. Здесь, следовательно, необходимо использовать приближенные методы.

Вводят понятие «профиль», который включает:

- мощность (количество полей) отношения R;
- «ширину» (число байт) любого атрибута;
- число различных значений атрибута в отношении R.

Используют метод оценки размеров промежуточных отношений для каждой операции реляционной алгебры.

Для решения вопроса о выборе копий и назначении узлов выполнения удобно строить граф оптимизации, который не содержит унарных операций, выполняемых в узлах. Операции декартова произведения и разности встречаются редко, чаще имеют место операции объединения и соединения.

Таким образом, задача оптимизации запроса отличается неоднозначностью решения.

Следует заметить, что задача оптимизации должна решаться регулярно и оперативно. Это требует от разработчика построения специальных программ и потому делается далеко не всегда.

Чаще отработанный алгоритм оптимизации закладывается при проектировании разработчиками СУРБД непосредственно в системе управления, при создании специальных программных продуктов в виде приложений или утилит.

12.2. ОДНОВРЕМЕННЫЙ ДОСТУП

Особенностью РБД является многопользовательский распределенный доступ к данным [11].

Сложность ее для РБД состоит в том, что механизм управления в одном узле не может в общем случае знать в каждый момент времени о взаимодействии с другими узлами. К тому же в узлах могут находиться и копии данных других локальных БД.

Взаимодействие элементов в РБД по-прежнему осуществляется посредством транзакции, которая становится распределенной (рис. 12.4) и, воздействуя на целостную БД, после своего завершения оставляет БД целостной. В транзакции выделяют две команды: читать (R) и писать-обновлять (W).

Здесь по-прежнему справедливы те же понятия, что используются в централизованных БД: синхронизация, расписание, последовательно-подобное расписание (ППР), конфликт, рестарт. Напомним, что две транзакции вступают в конфликт, если они работают с одним и тем же общим элементом данных и по крайней мере одна из них реализуется операцией «писать» (запись).

В СУРБД в каждом узле ведется свое расписание. Вектор расписаний всех узлов назовем *распределенным расписанием* (РР).

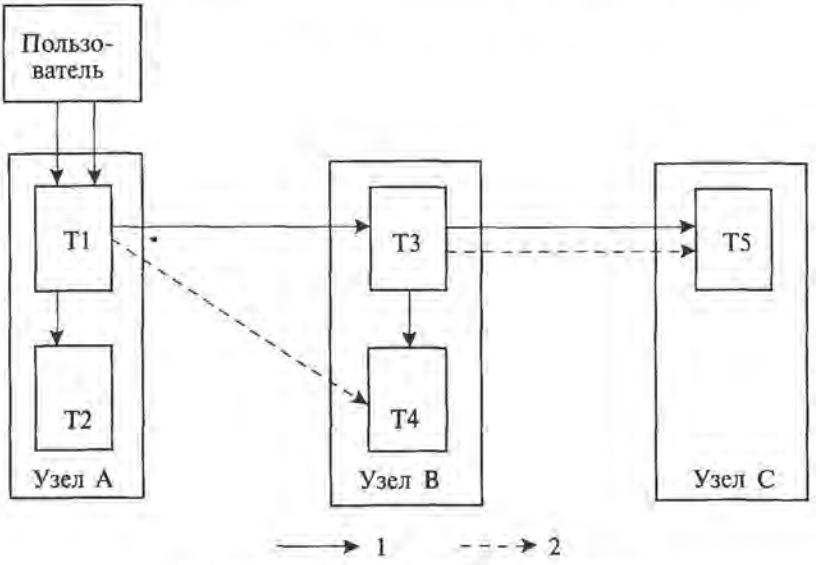


Рис. 12.4. Распределенная модель транзакции:
Ti — субтранзакции; 1 — порождающие транзакции; 2 — передача данных

Последовательное РР (ПРР) — это РР, в котором каждая составляющая последовательна. Последовательно-подобное РР (ППРР) эквивалентно ПРР.

Существует несколько методов синхронизации.

1. Блокировка, которая может быть: а) с главным узлом; б) с заданием блокировки с помощью предикатов; в) с главной копией.
2. Голосование по большинству.
3. Метод предварительного анализа конфликтов.

1. Блокировка

1. Выделенный узел становится приоритетным и управляет остальными узлами как в централизованной БД. Известно, что блокировка в централизованной БД предусматривает монопольный режим для записи и коллективный режим для чтения. Эти режимы используются и в блокировке с главным узлом, который руководит процессом синхронизации.

Применяют двух- и трехфазную транзакции. В двухфазной транзакции выделяют:

- расширение (блокировку ресурса);
- сжатие (возвращение ресурса и снятие блокировки).

Только главный узел выдает запрос на блокировку, который проходит по всей сети.

Действия по изменению данных допускаются только над предварительно заблокированными данными, при этом сначала должно быть проведено обновление всех копий. В таком режиме возможны тупики, которые могут устраняться снятием «младших» (позже поступивших) транзакций.

2. Любой кортеж t отношения r со схемой R, для которого предикат P(t) = true, не может быть вставлен в R, удален или изменен другой транзакцией до снятия блокировки.

Двухфазная блокировка работает здесь следующим образом.

В фазе расширения (подготовки) транзакция должна затребовать предикат-блокировку. Узел-координатор ведет таблицу блокировок. Блокировка осуществляется, если она не вступает в конфликт с другими блокировками. В противном случае транзакция переводится в режим ожидания или ей разрешается осуществить перехват ресурса.

В фазе сжатия (разблокировки) осуществляется удаление строки из таблицы блокировок: первая же разблокировка говорит о начале фазы сжатия.

Установление факта конфликта здесь по-прежнему проблематично: показано, что в общем случае проблема рекурсивно неразрешима.

Процедура установления факта конфликта определена математически для предикатов вида [атрибут|F|константа], где в качестве F используются операции из множества (<, =, >, ≠).

Тупиковые ситуации могут быть устранены в трехфазной транзакции (блокировка, выполнение, разблокировка).

Блокировки с главным узлом и с использованием предикатов сводятся к централизации процедуры синхронизации. В этом случае производительность БД ограничена возможностями своеобразного центра.

В свете сказанного дальнейшее развитие синхронизации пошло в направлении децентрализации.

3. В данном случае для любого элемента данных одна копия является главной, которую и блокирует транзакция в каждом узле. Очевидно, что имеется избыточность информации и ни один узел не является главным (децентрализованное управление).

2. Голосование по большинству. В общем случае предполагается избыточность РБД: в любом узле есть копия любого элемента.

Транзакция выполняется лишь в одном узле:

- команда чтения в своем узле работает без блокировок и синхронизации;
- при записи имя элемента и его новое значение заносится в список обновления.

После завершения транзакции список разносится по всем узлам и узел голосует за этот список (рис. 12.5).

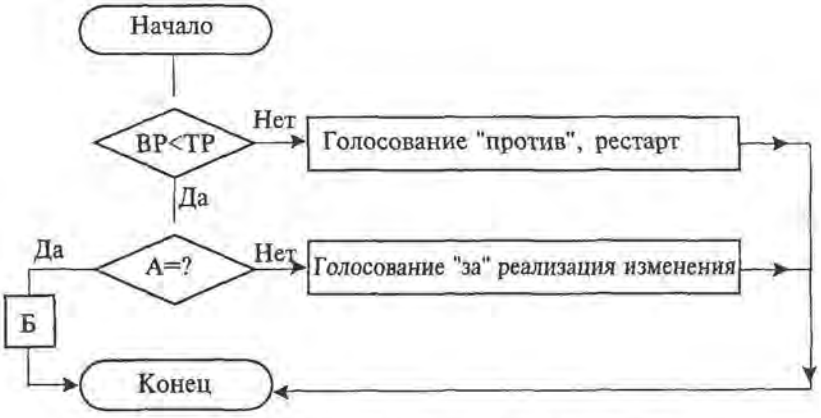


Рис. 12.5. Алгоритм голосования по большинству:

А — конфликт есть; Б — текущая транзакция получает бо́льшую временну́ю метку, чем ожидающая транзакция; ВР, ТР — временные метки последнего изменения и текущей транзакции

Если большинство — «за», транзакция принимается и обновление проводится во всех узлах.

Узел голосует «за», если:

1) элемент данных, прочитанных транзакцией, не был изменен после чтения;

2) транзакция T не конфликтует с другой транзакцией T′, которая ожидает решения (если данный узел проголосовал «за», но T′ еще не была принята или отвергнута в целом) в данном узле.

В позиции 1 используются временны́е метки (метки времени). Метка присваивается транзакции и каждому элементу данных (последней транзакции, которая его обновила).

Когда узел принимает список обновления, он сравнивает временны́е метки и определяет выполнение условий 1 и 2. Если условие 1 не выполняется, узел отвергает транзакцию, которая рестартует (рис. 12.5). Если условие 1 удовлетворяется, а 2 — нет, то узел не может голосовать «за» эту транзакцию, которая ожидает решения, пока его не получит. При выполнении обоих условий узел голосует «за».

Поскольку разные узлы получают списки обновления в разном порядке, они и голосуют в разном порядке, что может приводить к тупикам. Чтобы их избежать, узел голосует «против», если условие 1 выполняется, а 2 — нет и транзакция получает бо́льшую временну́ю метку (метку времени), чем ожидающая транзакция.

3. Метод предварительного анализа конфликтов. До сих пор синхронизировались все конфликтующие операции записи и чтения. Однако не все конфликты могут уничтожить последовательноподобие.

Определяются конфликты, которые действительно требуют синхронизации. Для этого выделяют классы, которые определяются множествами записей и чтения транзакций. Транзакция относится к данному классу, если множества чтения и записи ее принадлежат этому классу. С любым классом связан модуль транзакции (МТ) — компонента СУРБД, которая последовательно выполняет транзакции из этого класса. Тогда могут конфликтовать только транзакции разных классов, и их нужно синхронизировать. Конфликт классов моделируется графом конфликтов (рис. 12.6), вершины которого соответствуют множествам чтения и записи класса, а ребра представляют собой конфликты. Транзакции, не входящие в цикл, всегда последовательно подобны и не требуют синхронизации.

Граф конфликтов создается и анализируется статически, когда определены БД и классы. Классы, не входящие в цикл, помечаются как МТ и им сообщается о ненужности синхронизации.

Оставшиеся МТ должны синхронизировать свои транзакции.

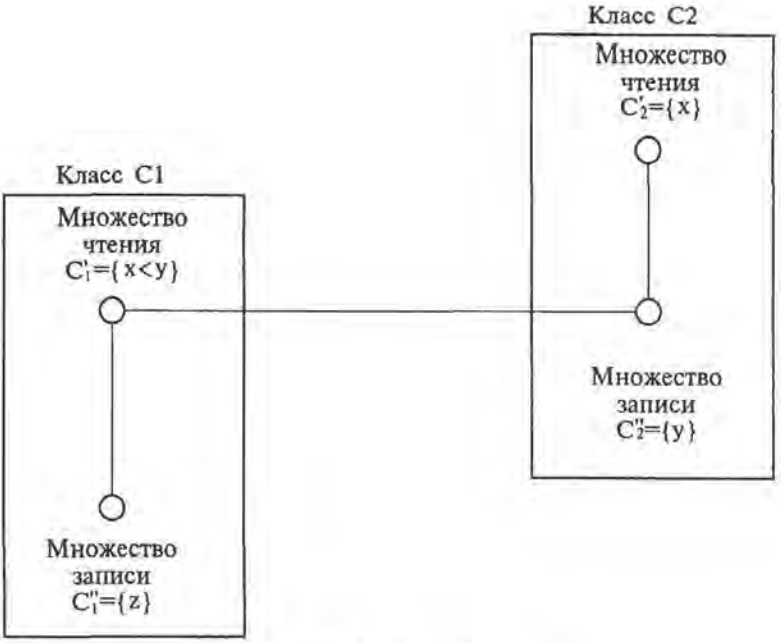


Рис. 12.6. Граф конфликтов

12.3. ЗАЩИТА ДАННЫХ, ВОССТАНОВЛЕНИЕ РБД

Здесь речь идет о защите данных в БД и приложений, запрашивающих данные.

Для защиты администратор РБД (АРБД) присваивает пропуска [11] для всех зарегистрированных в РБД пользователей и дает им разрешения на выполнение конкретных операций с конкретными данными. Проще всего это можно выполнить, как в СУРБД Oracle, с использованием языка SQL.

В таком случае, как и в централизованной БД, выделяют режимы запрещения и разрешения. К запрещениям относятся пароль, идентификатор пользователя. Они позволяют получить доступ к РБД в целом.

Отдельные части РБД могут иметь свою защиту. Для этого АРБД определяет полномочия (GRANT): перечень операций, которые пользователь может выполнять с соответствующими элементами данных.

Полномочия могут относиться к объектам или системе в целом. Полномочия для отдельных объектов («внутренность таблицы»)

предоставляют для какой-либо таблицы конкретным пользователям возможность выполнения операций (INSERT, UPDATE, DELETE). Системные полномочия касаются всей БД: создание, изменение структуры, удаление любой табличной области, опрос любой таблицы (CREATE, ALTER, DROP, SELECT).

Как и в централизованной БД, перечисленные полномочия могут быть отменены (REVOKE).

Для целей защиты АРБД может создавать (использовать) пароли.

В случае повышенных требований к защите данных следует использовать механизмы компьютерной безопасности, обеспечивающие конфиденциальность, целостность данных и защиту от отказов. Тремя главными механизмами защиты являются шифрование данных, контроль за процедурой загрузки и использование защитных криптографических контрольных сумм.

Если РБД обеспечивает целостные, непротиворечивые данные, говорят об ее корректном состоянии (корректности).

Восстановление (управление восстановлением) связано с приведением системы в корректное состояние после (аппаратного) сбоя. Любой конкретный метод восстановления реагирует на определенный отказ РБД.

В РБД возможны отказы узлов или сети связи. Поскольку над помехоустойчивостью сетей работают давно и успешно, чаще всего при рассуждениях считают сеть надежной. Воспользуемся пока этим предположением, считая ненадежными узлы.

Система восстановления решает две группы задач:

1) при незначительной неисправности — откат в выполнении текущей транзакции;

2) при существенных отказах — минимизация работы по восстановлению РБД.

Наиболее часто для целей восстановления используется системный журнал регистрации. При применении двухфазной транзакции вводится *точка фиксации*, когда принимается решение о фиксации транзакции. Для выполнения фиксации или отката используется информация журнала регистрации (блокировок, отмен и повторов).

Копирование изменений возможно после закрытия БД. Однако для РБД с интенсивными обновлениями копирование (архивацию) следует проводить в процессе работы РБД.

Журнал может содержать одну или более групп файлов регистрации и членов групп для физического сохранения изменений РБД. Любая группа включает один или более файлов, которые могут храниться на разных дисках, как это делается в СУБД Oracle.

Процедура восстановления проводится следующим образом.

1. Устраняется аппаратный сбой.

2. Восстанавливаются испорченные файлы данных путем копирования архивных копий и групп регистрации транзакций.

3. Запускается процесс восстановления:

а) с применением транзакций (применение к архивным копиям испорченных данных необходимых групп журнала транзакций);

б) с отменой незавершенных транзакций, оставшихся после восстановления с применением транзакций.

Нетрудно видеть, что процесс восстановления тесно связан с процедурой одновременного доступа (параллельного выполнения), который чаще всего организуется с помощью двухфазной транзакции. Условия работы РБД и система запоминания транзакций определяют систему восстановления пропущенных (отказавших узлов) транзакций и их последующее выполнение.

Изучим подробнее процесс отказа.

Возможно выделить следующие основные состояния узла: исправный, неисправный (застопоривший, неуправляемый), восстанавливаемый.

Застопоривший узел через некоторое время начинает исправляться и выполнять процедуру восстановления. Содержимое основной энергозависимой памяти может теряться, и восстановление идет за счет энергонезависимой стабильной памяти.

Неуправляемый узел ведет себя непредсказуемо и может выдавать странные сообщения, однако через некоторое время тоже начнет восстанавливаться.

Обсудим два возможных случая работы узлов в надежной сети: без дублирования и с дублированием данных.

При *отсутствии дублирования* не рассматриваются неуправляемые узлы (узлы с потерей управления).

Обычно одна транзакция при удаленном вызове (см. рис. 12.4) делится на серию субтранзакций. При откате одной из них осуществляется откат транзакции в целом: фактически система возвращается в точку фиксации. При этом невыполненные транзакции или субтранзакции и после устранения аппаратного сбоя выполняются вновь. Отметим, что в данном случае до устранения названного сбоя восстановить работу РБД фактически невозможно.

Резонно заметить, что откат всей транзакции не всегда нужен. Возможен другой способ: прервать только откатившуюся субтранзакцию, которая предпринимает фиксированное число попыток нового выполнения. Только если все они закончатся неудачей, осуществляется откат всей транзакции. Правда, здесь возможен перезапуск выполненных транзакций (каскадный возврат). Для его устранения, очевидно, нужен (рис. 12.4) информационный обмен (повторные сообщения) между транзакциями, т. е. усложняется процедура

одновременного доступа при одновременном упрощении процедуры восстановления.

При *дублировании* данных (разные копии хранятся в разных узлах) при застопорившем узле появляется возможность работы транзакций даже тогда, когда некоторые узлы находятся в неработоспособном состоянии.

Может быть частичное и полное (в каждом узле находится полная копия РБД) дублирование.

При *частичном дублировании* возможно, как и при параллельном доступе, использовать метод основной копии при отсутствии или при наличии активных узлов. Первый метод похож на рассмотренный метод главного узла. Один узел становится основным, по нему легко восстановить работу вспомогательных узлов. Транзакции могут быть пронумерованы. При восстановлении вспомогательный узел должен передавать основному узлу последний зарегистрированный номер транзакции (перед своим отказом) и затем запрашивать информацию об изменениях, проведенных к данному времени транзакциями с большими номерами.

Достоинствами метода основной копии являются единственная последовательность корректировки БД и уменьшение вероятности тупика.

В случае чтения во всех доступных узлах и использования активных узлов формируется и вводится список U записей об активных узлах, а основной узел используется для восстановления. «Основная» транзакция при обновлении должна запрашивать обстановку для записи и корректировки данных во всех узлах списка U, включая основной. Удается избежать некоторых конфликтов (чтение-запись, запись-запись). При наличии отказов и восстановлений список U меняется основным узлом и может возникнуть осложнение.

Пример 12.1. Пусть транзакция T формирует запрос на блокировку чтения в узле 1, а затем этот узел отказывает. Когда основной узел удаляет узел 1 из U, остальные транзакции теряют возможность запрашивать блокировку для записи в узле 1 и конфликт T и последующих транзакций может оказаться невыявленным.

Пример 12.2. Пусть узел 2 восстановился. Он запрашивает информацию о пропущенных транзакциях. Основной узел передает данные по транзакциям T_1, ..., T_n и добавляет узел 2 в список U. Однако транзакция T_{n+1}, находящаяся в состоянии фиксации, имеет старый список и не будет записывать в узел 2.

Таким образом, надо тщательно продумывать порядок работы, желательно с использованием временных диаграмм.

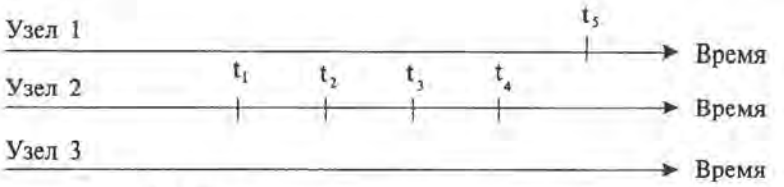


Рис. 12.7. Схема восстановления при потере управления:
t_1 — отказ; t_2 — выявление отказа; t_3 — начало восстановления; t_4 — конец восстановления

Возможны и другие методы:

1) выбор нового основного узла при отказе прежнего, при этом прежний узел должен быть восстанавливаемым, а новый — «видеть» обстановку, сложившуюся в прежнем узле. Доступ упрощается, но усложняется восстановление;

2) голосование по большинству: если откажут все узлы фрагмента, надо подождать восстановления мажоритарной группы.

Схема восстановления одного из узлов показана на рис. 12.7.

Пусть в момент t_1 отказал узел 2 и его БД разрушилась. Если узел 2 не будет исправлен до момента времени t_5, то отказ узлов 1 или 2 приведет к серьезному сбою. Узел 2 выявляет свой отказ к моменту t_2 и начинает восстанавливаться, используя «снимки» других узлов. На интервале $t_3 - t_4$ исправные узлы должны продолжать выполнение транзакций, которые должен запоминать узел 2 (и обрабатывать после момента t_4). С момента t_4 узел 2 присоединяется к остальным и система может выдержать второй отказ.

Контрольные вопросы

1. Какие критерии могут быть использованы для оптимизации запросов? Какой математический аппарат для этого применяется?
2. Каковы рекомендации по рационализации запросов?
3. Как используется операция полусоединения?
4. Каковы группы методов синхронизации в РБД?
5. Объясните суть блокировки с главным узлом и с использованием предикатов.
6. Что такое блокировка с главной копией: ее достоинства и недостатки?
7. В чем достоинства и недостатки голосования по большинству?
8. Опишите суть метода предварительного анализа конфликтов, его механизмы.
9. Что такое «восстановление» РБД?
10. Опишите процедуру восстановления.
11. Каковы возможности восстановления без дублирования? при частичном и полном дублировании данных?
12. Всегда ли нужен откат всей транзакции при откате какой-либо субтранзакции? Каков другой вариант?
13. Как можно использовать активные узлы?
14. Какие методы можно использовать при частичном дублировании? Можно ли при этом надежно устранить сбои в неуправляемых узлах?

Глава 13

Web-публикации баз данных

В последнее время все шире используют публикации БД в Web-страницы.

В данной главе рассмотрена технология публикации в рамках программной среды Delphi.

Показано, что для публикации используется многоуровневая структура режима клиент—сервер. Приведены основы языка HTML, используемого для публикаций. Приводится пример публикации базы данных (СУБД Paradox) с помощью среды Delphi. Результаты иллюстрируются экранными формами.

13.1. ОБЩИЕ ПОЛОЖЕНИЯ

Web-публикацией [2, 31, 32, 35] называется возможность поместить на Web-страницу динамические данные, которые имеются на сервере базы данных.

Цели публикаций могут быть различны: электронная торговля, связь с общественностью, реклама, образование. В данном случае в качестве цели выбрано обучение.

Схема реализации процедуры Web-публикации показана на рис. 13.1. Как нетрудно видеть, публикации реализуются с помощью варианта режима клиент—сервер с многоуровневой структурой (см. глава 11), в котором используется несколько отличная терминология.

Клиент (рис. 13.1) называется Web-браузером (иногда — просто браузером), сервер приложений — Web-сервером (www-сервером).

Запрос от браузера принимает и обрабатывает Web-сервер, реализующий поддержку протокола HTTP. Недостатком такой схемы является «статичность» (неизменность HTML-документа) передачи данных на Web-браузер.

Чтобы добиться динамичности публикации, на Web-сервере создают модули расширения (сценарии, скрипты — scripts). Задача скриптов (рис. 13.1) — принять и обработать запрос, извлечь из сервера БД нужные данные, которые представляются на языке про-

Рис. 13.1. Схема реализации процедуры Web-публикации

граммирования HyperText Markup Language (HTML), и передать обратно Web-серверу. Последний отправляет HTML-документ браузеру (клиенту).

Модули расширения могут быть написаны на языках программирования Java (JDBC, JSQL), JavaScript, VBAScript. Однако для этого надо знать хотя бы один из перечисленных языков.

Поэтому часто предпочтение отдается стандартным интерфейсам построения модулей расширения [2, 46—49]:

1) Common Gateway Interface (CGI);

2) ISAPI, NSAPI (Internet Services- и Netscape Server API).

Хронологически первым появился интерфейс CGI, однако он имеет существенные недостатки.

Первоначально скрипт представлял собой exe-файл, что для среды Windows было неудобно. Впоследствии перешли на вариант WinCGI, где параметры от сервера к скрипту передавались через INI-файл.

В то же время запускать каждый раз exe-файл означало потерю быстродействия. Для повышения быстродействия фирма Microsoft снабдила Internet-сервер протоколом ISAPI взаимодействия сервера с модулем расширения. В этом случае скрипт представляет собой динамическую библиотеку, загружаемую сервером.

Например, запрос find=petr браузера к библиотечному модулю test.dll сервера www.mysite.com может иметь вид

http://www.mysite.com.script/test.dll/find=petr

Работа с Web-страницами имеет свою специфику для различных приложений [46] и в общем случае ее описать трудно. В связи с этим сориентируемся на возможности приложения Delphi.

В рамках приложения Delphi [12, 25] ISAPI-модуль, охватывающий фактически Web-сервер и модуль расширения (рис. 13.1), мо-

жет реализоваться в виде модуля TWebModule, состоящего из модуля TDataModule и компоненты TWebDispatcher.

В публикации можно выделить две составляющие:

1) формирование Web-сервера;

2) собственно публикация.

Их удобно иллюстрировать на примере Delphi [32].

Пример 13.1

1. Создание Web-сервера с помощью ISAPI.

В этой составляющей выделяют в свою очередь создание источника (А), в качестве которого может выступать база данных, и формирование HTML-страницы (Б) на основе источника.

А. Создание источника данных. В меню File/New приложения Delphi выбрать шаблон WEB Server Application, в котором установить тип сервера ISAPI/NSAPI Dynamic Link Library. В инспекторе объектов вызвать редактор свойства Actions. Добавить с помощью свойства еще одну операцию Action, установить значения свойств PathInfo как /test, а Default как true. На закладке «События» инспектора объектов дважды нажать на мышь на событии OnAction.

В заголовке метода дописать

```
Response. Content:='Здравствуйте';
```

Это фактически содержание некоторой БД. Сохраним проект в виде

```
d:\...\InetPub\Scripts\IDAPITEST.DPR.
```

В дальнейшем источником будет служить база данных.

Б. Создание HTML-страницы.

Из этой страницы будет вызываться Web-модуль:

```
<HTML>
<HEAD>
<TITLE> Пример 13.1. </TITLE>
</HEAD>
<BODY>
<CENTER>
<BR> <A HREF=".../scripts/ISAPITEST.DLL/TEST"> Текст </>
</CENTER>
</BODY>
</HTML>
```

Сохраним эту страницу в d:\...\InetPub\WWWRoot\test.htm.

2. Осуществление публикации.

Запустить Web-браузер (например, через Internet Explorer). Набрать адрес http://localhost/test.htm. На экране монитора появится www-страница с подчеркнутым словом *Текст*. Если на него щелкнуть мышью, то на экране будет виден результат работы dll-библиотеки: слово *Здравствуйте*.

Возможен и прямой вызов dll-библиотеки по адресу

http://localhost/isapiproject.dll/test

Из примера 13.1 возможно сделать такие выводы.

1. Все процессы в публикации поддерживаются языком программирования HTML.

2. Приведен примитивный, иллюстративный пример.

Чтобы пример 13.1 стал достаточно реальным и удовлетворял пользователя, следует шире использовать возможности языка программирования HTML.

13.2. ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ HTML

Простейшая программа на языке HTML имеет следующий вид:

```
<HTML>
<HEAD>
<TITLE> ... </TITLE>
</HEAD>
<BODY>
...
</BODY>
</HTML>.
```

Из программы видно, что каждый блок, как правило, открывается оператором (**тегом**) <F> и закрывается тегом </F>.

Перечислим некоторые теги. Поскольку их достаточно много, введем некоторую классификацию [49].

Общее назначение:

<HTML> — программа на языке HTML;
<TITLE> — титул документа;
<HEAD> — заголовок страницы;
<BODY> — тело программы;
<p> — элемент абзаца;

 — переход на новую строку;
<NOBR> — выполнение в одну строку;
<PRE> — отформатированный текст.

Шрифт:

<B> — полужирный шрифт;
<EM> — курсив;
<U> — подчеркивание;
<BIG> — увеличение размера шрифта;
<I> — уменьшение размера шрифта;
<STRONG> — выделение текста;
<SUB> — нижний индекс;
<SUP> — верхний индекс;
<VAR> — переменная.

Списки:

<UL> <LI> </UL> — LI — нумерованный список;
<OL type 'I'> </OL> — где type имеет такие значения: I — I, II, III, ...; i — 1, 2, 3, ...; A — A, B, C, ...; a — a, b, c, ...

Рисунки:

<IMG>

Таблица:

<TABLE> — таблица;
<CAPTION> — заголовок таблицы;
<TR> — строка таблицы;
<TH> — заголовок столбца или строки внутри TR;
<TD> — ячейка таблицы.

Форма:

<form> — представляет интерактивную форму ввода и имеет два атрибута: method и action (действие). Чаще всего используют метод post, как способ передачи ISAPI-программе введенных данных. Например,

<form method=POST action=getphone.htm>.

<input> — поле ввода, задаваемое внутри тега <form>. Input имеет много атрибутов, из которых рассмотрим основные: name (название поля), value (значение поля), type.

Например,

<input type=text name=Familia value=Ivanov>.

В свою очередь, атрибут type тоже имеет несколько значений. Рассмотрим основные из них.

type=text — задает однострочные текстовые поля;
type= password — определяет пароль;
type=radio — задает радиокнопку;
type=checkbox — определяет флажок.
Например,

```
Select the credit card to place the order:<BR>
Visa<input type=radio name=CreditCard value=Visa>
Master Card<input type=radio name=CreditCard value=MCard>.
```

Type=range — проверка диапазона задания (от минимума до максимума);

Type=hidden — скрытые поля, которые используются обычно для формирования правил проверки. Например,

```
<form method=POST action=getphone.htm>
<input type=text name=WWWMY1 size=5>
<input type=hidden name=WWWMY1_range value='MIN=0
MAX=25'>
<input type=submit value='Ответ'>.
</form>
```

В последней строке submit — это кнопка, нажатие которой вызывает передачу введенных данных полей в Web-сервер, при этом value определяет название кнопки.

Аналогично задается кнопка reset для сброса значений полей к начальному состоянию.

Тег <select> отображает с помощью тега <option> некоторый список:

```
<select name=tovar>
<option> tea
<option> coffee
<option> water
</select>.
```

Для создания многострочного окна используют тег <textarea>.

13.3. РЕАЛИЗАЦИЯ ПУБЛИКАЦИИ

Теперь возможно рассмотреть пример публикации базы данных, т. е. процедур Б и 2А примера 13.1. Процедура 1А — создание источника данных — реализована в виде базы данных.

По-прежнему пример выполним в рамках приложения Delphi.

Для выполнения публикации удобно — для передачи базы данных на www-страницу — использовать специальные компоненты TDatasetTableProducer и TQueryTableProducer.

В частности, компонента TQueryTableProducer позволяет обращаться к базам данных на основе СУБД Access и InterBase, в которых таблицы размещаются в одном файле, через имя БД (файл) и SQL-оператор select с указанием в нем имени таблицы и публикуемых полей.

Для учебных целей составим простую программу с передачей полей таблицы в некотором цикле[1].

Исходная таблица построена в СУБД Paradox (рис. 13.2).

Она публикуется с помощью следующей программы.

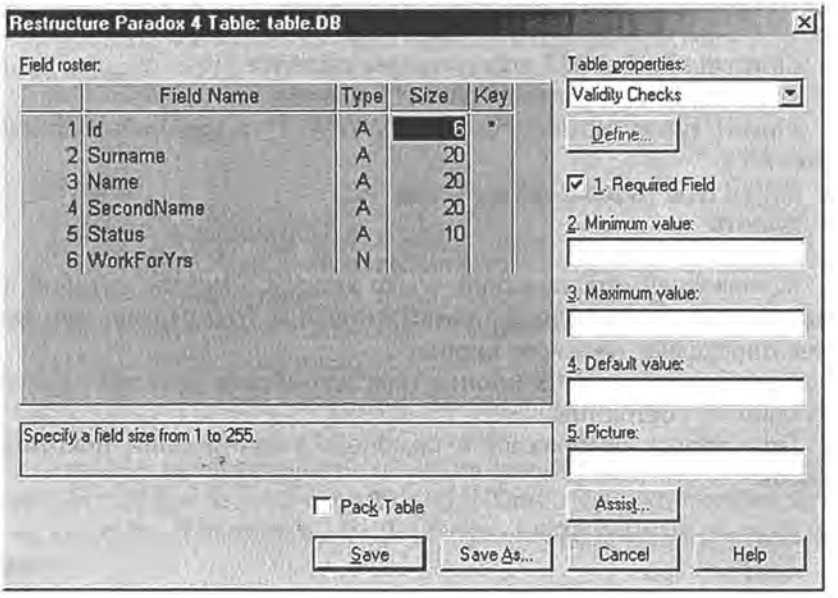


Рис. 13.2. Отображаемая таблица

[1] Работа выполнена С. Кузнецовым под руководством авторов.

```
unit Unit1;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls,
    Forms, Dialogs,
    OleCtrls, SHDocVw, Grids, DBGrids, Db, DBTables, HTTPApp,
    StdCtrls,
    Buttons, DBWeb;

type
    TForm1 = class(TForm)
        DataSource1: TDataSource;
        Table1: TTable;
        DBGrid1: TDBGrid;
        BitBtn1: TBitBtn;
        BitBtn2: TBitBtn;
        BitBtn3: TBitBtn;
        WebBrowser: TWebBrowser;
        procedure BitBtn1Click(Sender: TObject);
        procedure BitBtn2Click(Sender: TObject);
        procedure BitBtn3Click(Sender: TObject);
        procedure FindAddress;
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation

{$R *.DFM}
uses shellapi, filectrl; //подключение модулей

procedure TForm1.FindAddress;
var
    Flags: OLEVariant;
```

```
    begin
        Flags := 0; //установка адреса броузера

        WebBrowser.Navigate(WideString('c:\HTMLTEST.htm'), Flags,
        Flags, Flags, Flags);
    end;

    procedure TForm1.BitBtn1Click(Sender: TObject); //закрытие формы
    begin
    close;
    end;

    procedure TForm1.BitBtn2Click(Sender: TObject); //алгоритм пуб-
ликации

    var
    HTMLStr: TStringList;
    I,k: Integer;

    Begin //основные теги HTML
    HTMLStr:= TStringList.Create;
    HtmlStr.Clear;
    HtmlStr.Add ('<HTML>');
    HtmlStr.Add ('<HEAD>');
    HtmlStr.Add ('<TITLE>' + 'HTML-страница, содержащая отчет
из БД'+ '</TITLE>') ;
    HtmlStr.Add ('</HEAD>') ;
    HtmlStr.Add ('<BODY BGCOLOR=' #FFFFEE»>'); //добавление фай-
ла в заголовок //документа
    HtmlStr.Add ('<H1><CENTER>БД из файла '+ Table1 .TableName
+ '</CENTER></H1>');
    HtmlStr.Add ('<table border>');
    HtmlStr.Add ('<tr>');
    Table1.Open; //открытие таблицы для чтения
    for I:= 0 to Table1.FieldCount — 1 do
        HtmlStr.Add('<th>'+Table1.Fields[I].FieldName+'</th>');
        HtmlStr.Add('</tr>'); //закрытие заголовка таблицы
    Table1.First;
    while not Table1.EOF do //копирование данных из ячеек базы
данных в ячейки таблицы
        begin
        HtmlStr.Add("<tr>");
```

```
    for I:= 0 to Table1.FieldCount — 1 do
    if Table1.Fields[I].DisplayText=' then
    HtmlStr.Add('<td>' + '___' + '</td>')
    else HtmlStr.Add('<td>' + Table1.Fields[I].DisplayText + '</
    td>');
    HtmlStr.Add('</tr>');
    Table1.Next;
    end;
HtmlStr.Add('</tr>'); //закрытие выходного файла
HtmlStr.Add ('</table>');
HtmlStr.Add ('</BODY>');
HtmlStr.SaveToFile('c:\Htmltest.htm');
HtmlStr.Free;
end;

//запуск сгенерированного файла на просмотр
procedure TForm1.BitBtn3Click(Sender: TObject);
begin
FindAddress;
end;

end.
```

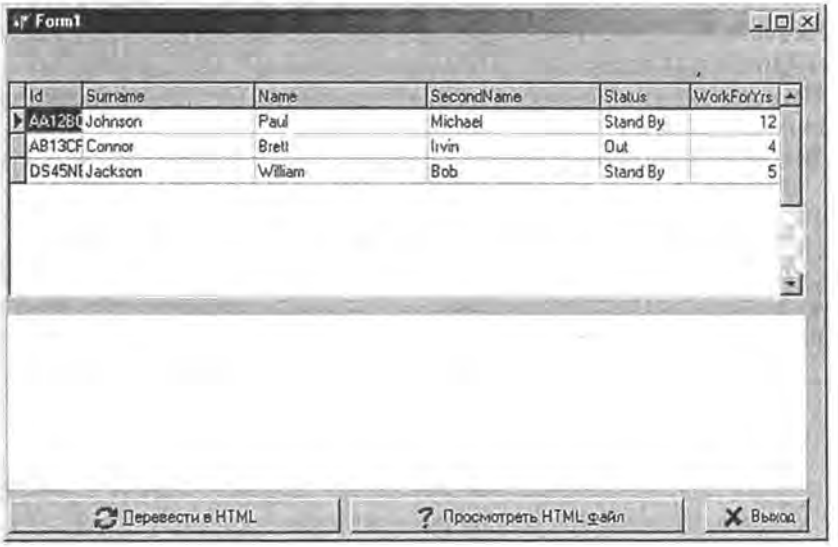


Рис. 13.3. Окно Delphi до запуска программы

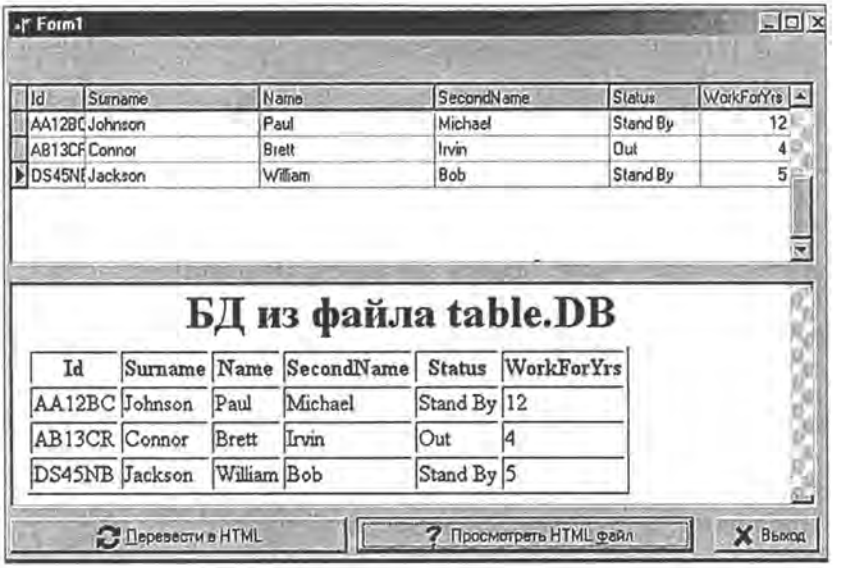


Рис. 13.4. Результат работы программы

Окно приложения Delphi до запуска программы показано на рис. 13.3. Кнопки формы особых комментариев не требуют.

После запуска программы и последовательного нажатия кнопок «Перевести в HTML» и «Просмотреть HTML-файл» в нижней части окна (рис. 13.4) отобразится сгенерированный HTML-файл.

При изменении данных в исходной таблице (рис. 13.4) меняется и результат в нижней части окна.

Выход из программы осуществляется нажатием кнопки «Выход».

Контрольные вопросы

1. В чем схожи схема Web-публикации и режим клиент—сервер?
2. В чем разница между статическим и динамическим HTML?
3. Зачем нужно расширение Web-сервера?
4. Какие интерфейсы расширения вы знаете?
5. Каковы составные части технологии публикации?
6. Как использовать компоненты Delphi для публикаций?

Глава 14

Проектирование и реализация баз данных

Рассмотрим вопросы создания и реализации БД, которые приведены в гл. 14 и 15, словесное описание дано в гл. 1 (примеры 1.1 и 1.2).

Речь пойдет о наиболее распространенных реляционных операционных базах данных [27—37]. Создание объектно-ориентированных баз данных имеет свою специфику [38] и в настоящей работе не рассматривается.

Напомним, что эти примеры характеризуют соответственно традиционный и современный подходы к созданию БД.

В данной главе детально обсуждается пример 1.1 гл. 1 с его реализацией в двух вариантах: в СУБД Access и в СУБД InterBase с использованием программного продукта Delphi.

Первый, локальный, вариант предназначен для начинающих пользователей, второй, сетевой, — для «продвинутых» пользователей, знающих основы программирования хотя бы в рамках языка Pascal.

Второй вариант служит как бы полигоном для лучшего понимания последующей реализации примера в гл. 15.

14.1. ПРОЦЕДУРА ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Сразу следует отметить, что процедура проектирования БД, как и любая процедура проектирования, характеризуется большим количеством неформальных факторов и потому неоднозначна. В данной работе приводится один из возможных вариантов проектирования.

Последовательность этапов проектирования показана на рис. 2.6. Рассмотрим этапы более подробно.

Анализ требований. На начальной стадии развития БД считались автономными, поскольку предполагалось построение универсального хранилища данных. Полагали, что все исходные данные известны. В силу этого на данном этапе велся лишь анализ возможностей реализации требований, предъявляемых к проектируемой базе данных.

Вскоре выяснилось, что БД является частью более сложной системы, чаще всего — АСУ, поэтому исходные данные следует получать при изучении документооборота или при выявлении алгоритмов приложения базы данных.

Первый этап получил более точные названия [1—6], наиболее удачным из которых, по мнению авторов, является [2] «планирование базы данных».

При этом выявились [4] два подхода к проектированию БД:

1) традиционный (классический), сформировавшийся в 80-е годы XX века и применяемый до сих пор в информационно-поисковом режиме АСУ;

2) современный, формирование которого началось в 90-е годы и продолжается до настоящего времени с использованием в информационно-советующем режиме систем поддержки принятия решений.

Первый подход преследовал цель автоматизации документооборота. Задача ставилась так. На входе БД имеется одна система документов (таблиц), данные для которых получались из разных источников. База данных преобразует систему входных документов в систему выходных документов, удобных для работы конечного пользователя. Алгоритм преобразования, когда это не вызывает разночтений, будем называть в общем случае алгоритмом приложения.

Второй подход исходит из решения задачи [44], т. е. от алгоритма приложения, на основе которого и создается база данных. Под **приложением** понимается программа или группа программ, предназначенных для выполнения определенных однотипных работ.

Изменение парадигмы построения БД связано с тем, что построение универсальной базы данных себя не оправдало. Расширение сферы применения объектно-ориентированного подхода и его гибкость создали широкие возможности для интеграции алгоритма приложения и собственно базы данных.

Во втором подходе сформировалась следующая технология.

1. Определение цели проектирования.

2. Выявление логики приложения.

3. Планирование схемы связей приложения (системы таблиц), иногда называемой DLL-сценарием.

Первый этап второго подхода имеет другую направленность — построение алгоритма приложения и системы данных для него.

В любом подходе результатом первого этапа является техническое задание (ТЗ), в котором фиксируются требования, предъявляе-

мые к БД. ТЗ является основой для работы на последующих этапах проектирования.

Концептуальная модель. Целью следующего этапа является построение — на основе ТЗ — основных положений создаваемой БД без привязки к конкретной СУБД.

Зарубежные проектировщики предпочитают [2, 3, 5] использовать для этих целей ER-диаграммы, несомненным достоинством использования которых является возможность автоматизации проектирования БД.

Вместе с тем при использовании ER-диаграмм возникают серьезные сложности:

- с ER-диаграммами может работать профессионал высокого уровня, по-видимому, из-за недостаточной наглядности диаграмм при наличии значительного количества неформальных факторов;
- в аппарате ER-диаграмм фактически отдельно строятся диаграммы сущностей и атрибутов, что затрудняет указание полей (атрибутов) связи сущностей;
- нет четкой привязки сущностей к входам и выходам системы, составной частью которой является проектируемая БД;
- не очень четко просматривается такая привязка и в DF-диаграммах, сопровождающих ER-диаграммы.

Нечеткое выявление сущностей может вызвать серьезные ошибки [6] на начальных этапах проектирования. Такие ошибки впоследствии очень тяжело исправить.

В то же время в системном анализе четко выявляются входы, выходы, потоки информации, их формы и характеристики.

В связи с этим при первом знакомстве с БД полезнее, по мнению авторов, использовать более наглядные схемы связей. После приобретения определенного опыта возможно постепенно перейти к использованию ER-диаграмм.

На основе ТЗ выбирается режим (одно- или многопользовательский) и разновидность (централизованная или распределенная) БД.

В конце этого этапа осуществляется — опять на основе ТЗ — выбор СУБД, который, как показано в гл. 9, предполагает выбор как модели данных, так и СУБД в рамках выбранной модели.

На данном этапе возможна и процедура нормализации.

Простейшая технология нормализации (ТН1) заключена в следующем.

1. Выписать в строку (линейно) перечень всех полей, полученных на этапе анализа требований.

2. Выявить связи между полями.

3. Провести процедуры нормализации с построением 1НФ—5НФ.

Однако при большом количестве полей эта процедура трудоемка, поэтому целесообразно использовать технологию ТН2.

1. Разделить поля на блоки (по предметному признаку).

2. Установить связи между блоками.

3. Провести нормализацию на уровне выделенных блоков.

При необходимости на последующем этапе выполнить нормализацию внутри блоков.

При однопользовательском режиме перечень полей определяется схемой (данных). При многопользовательском режиме возможно использование одного из двух вариантов формирования схемы.

1. *От схемы — к подсхемам.* На основе изучения документооборота и алгоритма приложения (алгоритма преобразования) формируется схема, которая далее «разбрасывается» между пользователями.

2. *От подсхем — к схеме.* Для каждого пользователя выявляется подсхема, из которых составляется общая схема [1—3, 39].

В настоящее время данные извлекаются на этапе анализа требований, и потому чаще используют первый вариант.

Описанные варианты могут использоваться и на этапах фрагментации и локализации при построении распределенных баз данных.

Фрагментация и локализация. Первоначально проводится горизонтальная фрагментация, в которой не предусматривается дублирование данных. Выявляются узлы, в которых решаются соответствующие задачи алгоритма преобразования (алгоритма приложения). Желательна высокая степень локализации данных в узлах. Дублирование осуществляется чаще всего в целях повышения надежности, реже — для повышения быстродействия. Предпочтительно использование режима клиент—сервер при создании одноранговой сети только между серверами.

Логическая модель. Здесь проводят окончательную нормализацию данных. На этом этапе учитываются характеристики выбранной СУБД, в частности система типов данных, методы обеспечения целостности данных, способы защиты и восстановления данных.

Физическая модель используется в случае выбора иерархической СУБД. При работе с реляционной СУБД физическое моделирование сводится к выбору размеров страниц (Delphi), областей данных (Oracle) и применяется, как правило, при построении сверхбольших (по объему) БД. Во всех остальных случаях используют «физические параметры» по умолчанию.

14.2. ПРОЦЕДУРА РЕАЛИЗАЦИИ БАЗ ДАННЫХ

В реализации БД, как отмечалось ранее, выделяют «компьютерное» создание собственно БД, интерфейса пользователя и алгоритма приложения (алгоритма преобразования).

Создание **собственно БД** предполагает использование языка программирования SQL или QBE для:

- построения структуры таблиц (задание имени таблицы и перечня полей с соответствующими типами данных, указание ключей, индексов и ограничений на отдельные поля или группу полей);
- установление связей между таблицами с учетом ссылочной целостности;
- заполнение таблиц данными.

В заполнении таблиц выделяют: начальное заполнение — для тестовой проверки работы БД; рабочее заполнение. Заполнение возможно осуществлять вручную (удобнее всего для небольших по объему таблиц и при начальном заполнении) или программно (при заимствовании данных из других электронных источников).

Ручное заполнение возможно двумя способами:

- непосредственно самой таблицы (в том числе и через соответствующие утилиты);
- через специально создаваемые формы, относящиеся к интерфейсу пользователя и являющиеся в какой-то мере аналогом строки «бумажной» таблицы вдоль линейки, положенной на эту строку таблицы.

Интерфейс пользователя. Под интерфейсом пользователя [45] понимается совокупность информационной модели (ИМ) проблемной области, средств и способов взаимодействия пользователя с этой моделью, а также компонентов, обеспечивающих формирование ИМ в процессе работы программной системы. Интерфейс играет важнейшую роль в «приживаемости» разработанных программных продуктов.

Отметим, что задача проектирования интерфейса пользователя *многовариантна*, при этом процедура выбора в значительной степени неформальна. В силу этой неформальности следует упорядочить процедуру проектирования введением набора правил-принципов. Перечислим основные из них.

1. Ориентация на требования пользователя (а не на умение разработчика) — User_centred Design.

2. Согласованность (использование однозначных команд в рабочей среде).

3. Наличие обратной связи от компьютера (сигнал о восприятии команды).

4. Простота интерфейса — предоставление только тех его элементов управления, которые нужны на данном шаге сеанса работы с программным продуктом. Под *сеансом* (*сессией*) понимается интервал времени от запуска продукта до выхода из него.

5. Гибкость интерфейса — способность учитывать уровень подготовки пользователя.

6. Учет эстетических требований, в которые включаются время освоения интерфейса, время решения задачи, субъективная удовлетворенность удобством интерфейса.

7. Учет традиций предметной области.

8. Итерационный характер разработки интерфейса с учетом предпочтений пользователя.

9. Учет возможностей программных и аппаратных средств.

10. Возможность учета психологических характеристик пользователя: левое полушарие мозга лучше воспринимает текст, правое — образы, изображения.

11. Учет возможностей упрощения программирования.

12. Малое время отклика компьютера: если интервал от запроса до ответа компьютера превышает 20 с, систему не считают интерактивной.

Перечисленные требования порой противоречивы, однако их следует учитывать.

Основой интерфейса пользователя является система элементов управления совместно с формами и отчетами.

В качестве синхронизирующего объекта в интерфейсе пользователя используют либо (головное) меню, либо кнопочную форму.

Алгоритм приложения. В традиционном подходе формируется алгоритм прямого преобразования данных исходных таблиц в выходные документы. В качестве выходных документов (таблиц) выступают запросы и отчеты. Запросы формируются с помощью языка программирования SQL или QBE. Отчеты являются фактически запросами, снабженными дополнительными пояснениями. Отчеты могут реализоваться с помощью различных конструкторов (при фактически ручном создании) или с помощью мастеров (в полуавтоматическом режиме).

Алгоритмы приложения в современном подходе отличаются от алгоритмов приложения более сложными вычислениями с возможным формированием промежуточных объектов. Алгоритм приложения может быть написан на различных языках программирования — например, Visual Basic for Applications (VBA), Object Pascal, SQL. В последнем случае он выполняется в виде хранимых процедур, генераторов, триггеров.

В заключение следует отметить, что процесс проектирования итеративен: могут постоянно изменяться требования в ТЗ, составляющие проектирования и реализации БД.

Процессы проектирования и реализации рассмотрим на двух примерах — «Учебный процесс» и «Прием на работу», сжатое описание которых приведено в примерах 2.1 и 2.2.

Реализация описанных в примерах баз данных проводится на СУБД Access и InterBase (в рамках программного продукта Delphi).

Многочисленным возможностям указанных СУБД посвящены специальные отдельные публикации [27—36, 41]. Освоение возможностей, как показал опыт, целесообразно проводить в два этапа:

1) рассмотрение основной ветви технологии реализации на конкретном примере (БД) с использованием ограниченного круга возможностей СУБД;

2) детальное освоение — по подробным описаниям [27—36, 41] — названных СУБД при учете их многочисленных частных возможностей.

Работы на втором этапе после тщательного изучения технологии на первом этапе, как показывает практика, не вызывает особых затруднений. В то же время описание многочисленных частных процедур второго этапа, как явствует из работ [17—36], занимает объем, превышающий настоящую публикацию. В связи с этим в данной работе осветим только первый этап.

Процедура создания хранилищ данных только начала формироваться. В настоящее время выделились два варианта:

- полный, описанный в гл. 2;
- «усеченный», заключающийся в периодической передаче в архивные таблицы данных из оперативных таблиц. Связи между архивными таблицами чаще всего отсутствуют.

Второй вариант приведен в работе [32] и рассмотрен в § 15.4 данной работы.

14.3. ЦЕНТРАЛИЗОВАННЫЕ БАЗЫ ДАННЫХ

Проектирование централизованной БД

Рассмотрим БД «Учебный процесс».

Анализ требований. Учебный процесс сопровождается в деканате значительным потоком информации, которая до сих пор обрабатывалась или вручную, или с минимальным использованием компьютерной техники.

Трудности процесса обработки определяются тем, что ее необходимо провести качественно (без ошибок) и в очень сжатые сроки

(начало и конец осеннего семестра, конец весеннего семестра). По результатам обработки необходимо не только принять информацию к сведению, сформировать ряд документов (справки, ведомости), но и выработать решения (подготовить приказы об отчислении, переводе студентов на следующий курс).

Работа деканата факультета характеризуется значительными потоками документальной информации, связывающими деканат практически со всеми подразделениями института.

Одна и та же информация может присутствовать в различных документах. Например, список студентов — в приказах ректора, в зачетных и экзаменационных ведомостях, в списках студентов, подлежащих призыву, в документах договорного отдела, в распоряжениях деканата.

Изменение хотя бы одного элемента данных может привести к необходимости синхронного изменения данных в целом ряде документов.

Существующая обработка данных при дефиците времени чревата следующими возможными последствиями.

1. Низкая оперативность подготовки необходимой документации из-за значительной трудоемкости процесса.

2. Ошибки в заполнении документов из-за дефицита времени. Так, процесс занесения оценок в карточку выполнения учебного плана занимает несколько дней. Различие в формах (структуре) карточки и экзаменационной ведомости, как источника первичной информации, может привести к дополнительным ошибкам введения данных.

3. Ручное дублирование элементов информации в различных документах разных подразделений может привести к разнобою в одних и тех же данных (нарушению целостности данных).

Сказанное позволяет сделать вывод о необходимости построения компьютерной базы данных учебного процесса.

Для формирования состава полей БД проведем обследование учебного процесса (имеющегося документооборота).

На факультете четыре кафедры, три из них — выпускающие. Студенты разделены на группы: за каждой выпускающей кафедрой закреплено по 6 групп.

Учебный процесс цикличен, и поэтому достаточно рассмотреть лишь один (годовой) цикл, состоящий из осеннего и весеннего семестров.

В конце весеннего семестра (апрель—май) учебного года на основе учебного плана, поступающего из учебного отдела, деканат составляет учебные графики. Они являются фактически скоррек-

тированными годовыми «вырезками» из пяти- или шестилетних планов соответствующих специальностей. Это — основной источник информации о составе учебного процесса.

В течение учебного года деканат организует выполнение студентами и преподавателями учебного графика.

В конце семестров (декабрь—январь, май—июнь) деканат выдает ведомости (зачетные и экзаменационные), в которые преподаватели кафедр вносят оценки об экзаменах, зачетах, курсовых проектах (работах). Полученные данные заносятся в журнал «Выполнение учебного плана» и в «Учетную карточку студента».

Далее начинается обработка карточек по следующим алгоритмам.

1. Успевающим студентам начисляется стипендия. Список таких студентов передается в канцелярию (приемную ректора) и в бухгалтерию.

2. Студенты, не ликвидировавшие задолженности (одна или более неудовлетворительных оценок), представляются к отчислению. Соответствующие списки вместе с проектом приказа передаются в канцелярию и коммерческий отдел.

3. Только в весеннем семестре готовится приказ о переводе успевающих студентов на следующий курс, об успешном окончании вуза дипломниками и их отчислении из института.

Схема «ручных» информационных потоков учебного процесса представлена на рис. 14.1.

Таким образом, формируется перечень «минимальных» необходимых выходных документов (запросов и отчетов).

Перечислим основные из них.

1. Выдача списков студентов по группам (с указанием фамилий старост).

2. Поиск фамилий студентов (по группам), сдавших сессию на «отлично», с одной оценкой «хорошо», с двумя оценками «хорошо», подготовка и распечатка соответствующих приказов на стипендию.

3. Формирование зачетно-экзаменационных ведомостей.

4. Выдача списка преподавателей по кафедрам с указанием предметов, преподаваемых ими в данном семестре.

5. Формирование в конце учебного года приказов о переводе на следующий курс успевающих студентов и об отчислении неуспевающих студентов (с распоряжением и представлением декана соответственно).

6. Выдача списка военнообязанных студентов.

Система полей (столбцов) таблиц базы данных «Учебный процесс» выявляется двумя способами: в результате бесед с пользователем и на документальной основе (по номенклатуре дел деканата).

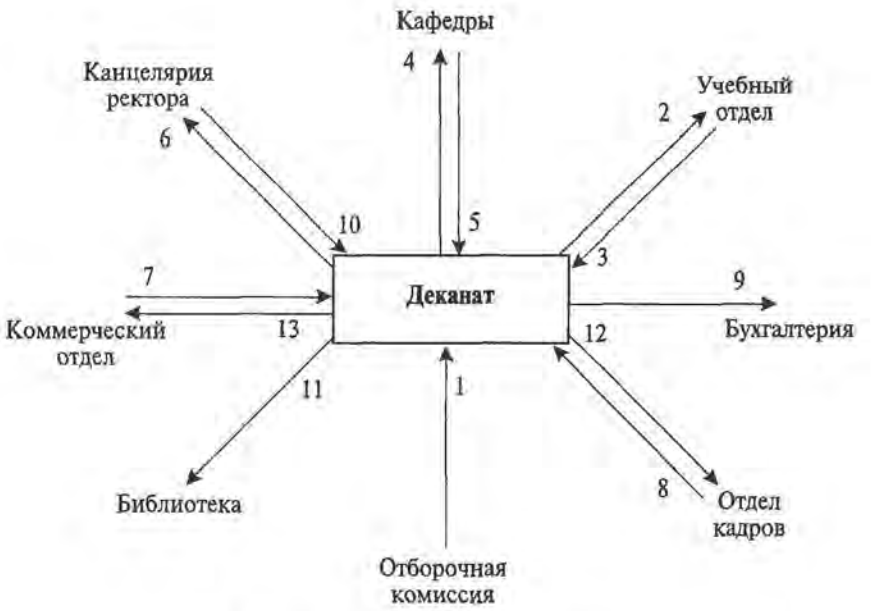


Рис. 14.1. Документооборот деканата:

1 — личные дела, приказы на зачисление; 2 — численность групп, учебный порядок; 3 — расписание семестра; 4 — приказы ректора, распоряжения деканата, ведомости; 5 — ведомости, список преподавателей, список кураторов; 6 — проекты приказов на студентов, академических справок, приложений к диплому; 7 — список оплачивающих обучение, изменивших форму оплаты; 8 — список призывников, список живущих в общежитии; 9 — приказы на стипендию, отчисление, различные виды помощи; 10 — приказы ректора, академические справки, приложение к диплому; 11 — список групп; 12 — списки преподавателей; 13 — списки принятых на 2—6 курсы

В силу значительной номенклатуры документов выделим основные из них и классифицируем. Определим четыре класса.

1. *Документы, возникающие в деканате и там используемые (внутренние документы).*

03. Положение о факультете.

07. Указания и распоряжения декана.

16. Отчеты об итогах экзаменационной деятельности и внутрисеместровой аттестации.

22. Личные карточки студентов (форма 13).

23. Учетные карточки студентов (форма 14).

27. Журнал выдачи студенческих билетов и зачетных книжек.

2. *Документы, формируемые в деканате и передаваемые в другие подразделения.*

08. Годовой план и отчет факультета о работе в учебном году.

09. Календарный план работы деканата.

14. Статистический отчет о движении студентов за учебный год.

20. Материалы по отчислениям.

24. Списки студентов по учебным группам (форма 18).

3. *Документы, поступающие из других подразделений и используемые только в деканате.*

04. Приказы и указания ректора по основной деятельности.

05. Приказы и указания ректора по личному составу студентов.

17. Материалы по производственной практике.

18. Экзаменационные и зачетные ведомости.

19. Журнал результатов экзаменационных сессий и аттестаций.

4. *Документы, поступающие из других подразделений, обрабатываемые в деканате и передаваемые в другие подразделения.*

02. Типовые и учебные планы.

13. Отчет о выполнении нагрузки.

15. Дипломное проектирование.

16. Экзаменационные и зачетные ведомости.

25. Расписание учебных занятий и экзаменационных сессий.

Наибольший интерес для построения БД представляют дела 02, 05, 07, 08, 13, 14, 16, 17, 20 и особенно 22—25.

Покажем процедуры обработки информации при использовании БД. Для этого детализируем рис. 14.1 и получим рис. 14.2. Алгоритм приложения этого рисунка детализируется на рис. 14.3. Из него следуют три алгоритма обработки информации.

Алгоритм А.1. Составить список студентов, сдавших сессию только на «отлично». Определить список студентов, сдавших сессию не ниже, чем на «хорошо». Составить проекты приказа на повышенную и обычную стипендии.

Алгоритм А.2. Составить список студентов, имеющих хотя бы одну оценку «неудовлетворительно». Сформировать проект приказа на отчисление.

Алгоритм А.3. Составить список студентов, не имеющих ни одной неудовлетворительной оценки. Представить проект приказа на перевод на следующий курс.

На основании представленных сведений о документообороте сформируем основные требования к БД в виде технического задания.

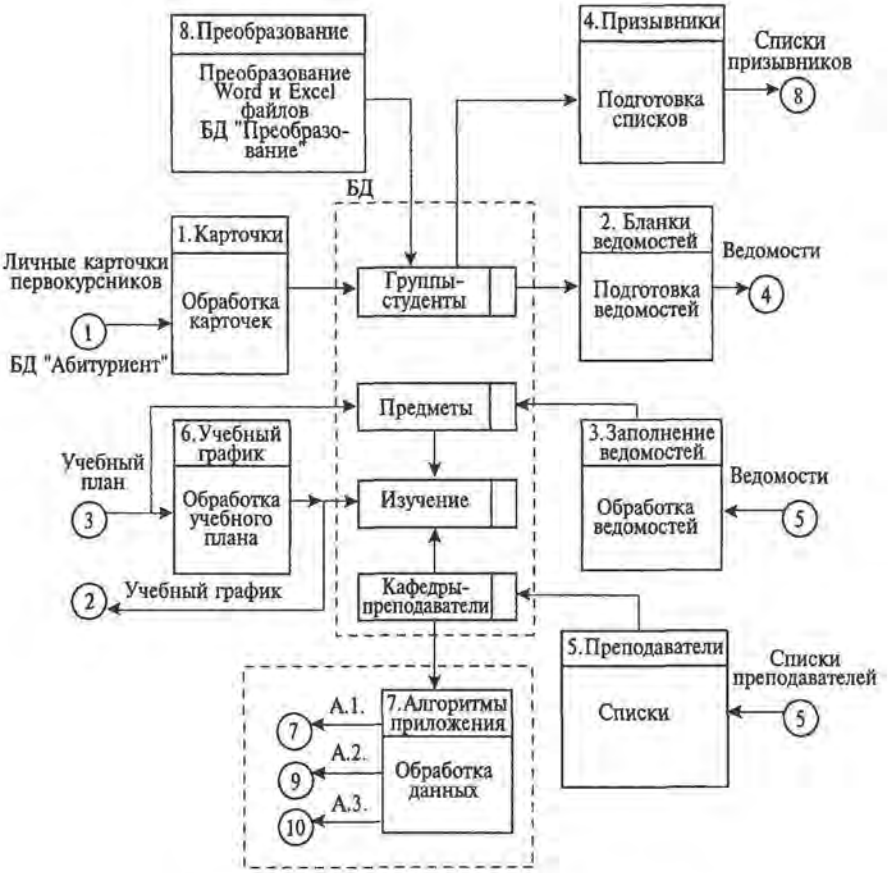


Рис. 14.2. Схема основных потоков информации (обозначения см. на рис. 14.1)

Техническое задание

К создаваемой БД следует предъявить такие общие требования.

1. БД должна быть рассчитана на пользователя — непрофессионала в программировании.

2. БД следует строить на основе широко распространенной СУБД.

3. Объем БД будет постепенно наращиваться и на начальном этапе не будет превышать 1 Гбайт.

4. БД должна иметь удобный интерфейс.

5. В БД должны быть предусмотрены возможности запросов произвольной структуры при работе с БД на рабочем месте.

6. В базе данных находятся исчерпывающие данные о студентах.

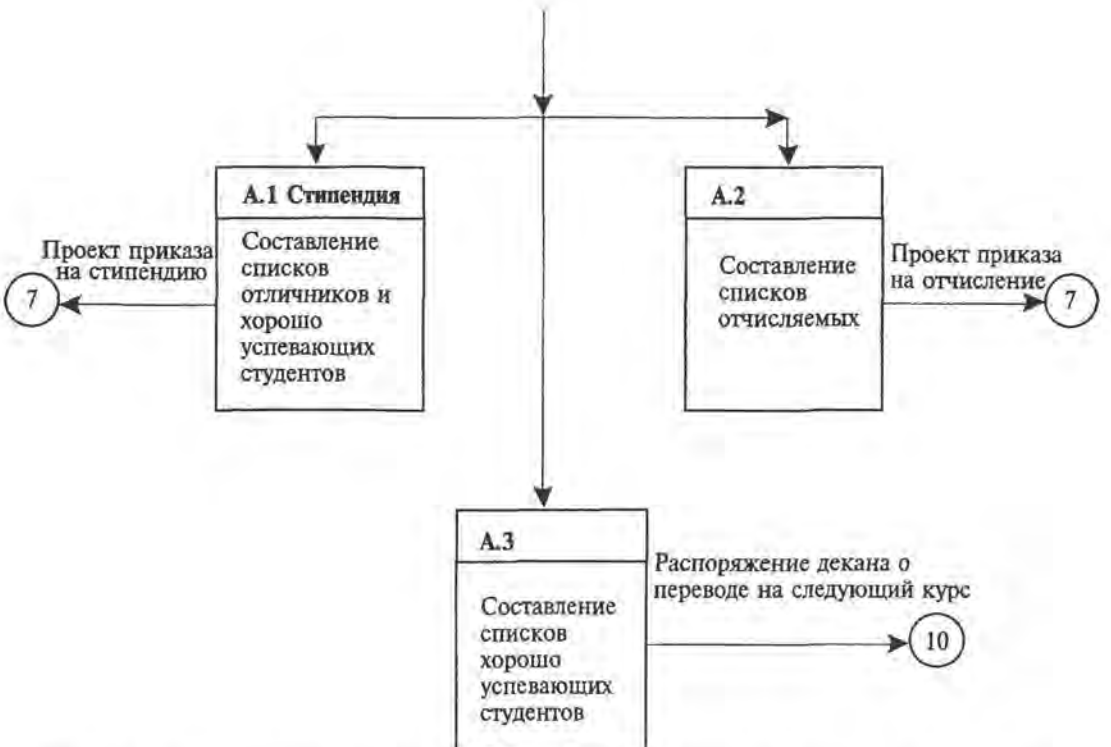


Рис. 14.3. Детальная схема потоков информации. Обозначения те же, что и на рис. 14.1. и 14.2

Необходимо предусмотреть вопросы защиты информации. Данные для базы данных предоставляет заказчик.

В базе данных на первом этапе следует предусмотреть следующие таблицы, формы и отчеты.

Таблицы и их поля приведены позднее в виде схемы связей.

В каждой таблице на первом этапе должно быть не менее 10 записей, вводимых исполнителем. Предусмотреть ввод записей прямо в таблицы или с помощью соответствующих форм.

Отчеты

Ведомость.
Группа.
Преподаватель.
Студент.
Список военнообязанных.
Приказ о переводе.
Результаты сессии.
Задолжники.
Отличники.

Формы

Предмет.
Группа—студент.
Кафедра—преподаватель.
Специальность.
Изучение предметов.
Успеваемость.
Главная кнопочная форма.

Один раз в год следует изменять ряд данных (название группы, номер курса) в полуавтоматическом режиме. Шестой курс в этом случае становится на один год как бы седьмым курсом.

В БД должны храниться в течение одного года данные о выпускниках («группы 7-го курса»). Кроме того, должны храниться данные о более ранних выпусках. Частота обращения к этим архивным данным гораздо ниже, чем к «рабочей» информации. В то же время эти данные могут излишне загружать БД, снижая ее быстродействие. Во избежание такого нежелательного эффекта архивные данные лучше в полуавтоматическом режиме передавать в другую, архивную базу данных, которая в теории баз данных получила название «хранилище данных».

Значительная часть данных для БД «Учебный процесс», содержащаяся в БД «Абитуриент», должна считываться оттуда в полуав-

томатическом режиме. Не менее объемная часть данных должна поступать из учебных планов и графиков. При трудоемком ручном вводе подобной информации может возникнуть множество ошибок. Чтобы уменьшить их количество, следовало бы автоматизировать процесс составления названных планов и графиков, предусмотрев полуавтоматический процесс ввода информации в БД «Учебный процесс».

Необходимо предусмотреть автоматический подсчет некоторых характеристик (например, число студентов в группе после изменений в ней — отчисления или добавления студентов в группу).

Для отчисленных студентов следует, видимо, создать специальную таблицу, куда в автоматическом режиме передавать данные при отчислении студента и «возвращать» эти данные при его восстановлении. Возможно, такой же вариант следует предусмотреть для студентов, находящихся в академическом отпуске.

Хранение данных об уволившихся преподавателях проводится в архивной БД.

ТЗ служит основой для работы на последующих этапах.

Концептуальная модель БД. Документальные данные и данные, полученные в результате устных бесед с сотрудниками деканата, возможно с точки зрения построения БД классифицировать на следующие виды:

1) группа (студенческая);
2) предмет (дисциплина);
3) кафедра;
4) предприятие (место практики).

Они связаны соотношением М : М. Такое отношение не реализует ни одна модель данных. Следует перейти к набору отношений 1 : М. В результате появляются новые группы данных: студент, преподаватель, ИПП.

По возрастанию частоты обновления данных возможно выделить такой порядок: кафедра, предмет, преподаватель, группа, ИПП, студент.

Данные о кафедре и предмете обновляются обычно раз в несколько лет. Регламентное обновление данных о преподавателе, группе осуществляется обычно через год. Данные об ИПП меняются раз в семестр. Различные данные о студенте меняются с разными интервалами: год (студенты вновь принятых групп), семестр (данные о зачетах и экзаменах), более короткие интервалы (дополнительное зачисление, отчисление, оплата обучения). Регламентная частота считывания практически всех видов данных — семестр, хотя для отдельных видов данных интервал считывания значительно короче.

По убыванию размерности данных виды можно расположить в таком порядке: студент, предмет, ИПП, преподаватель, группа, кафедра.

Выбор СУБД. Поскольку речь идет об обработке информации на рабочих местах с помощью персонального компьютера, следует выбрать реляционную модель данных, наиболее широко используемую для названных целей. В соответствии с ТЗ и возможностями приобретения наиболее подходят СУБД FoxPro и Access. Из-за более удобного интерфейса и большей распространенности предпочтение отдается СУБД Access-2000.

Приведем основные предельные возможности СУБД Access, необходимые для компьютерной реализации примера.

Объем БД — не более 1 Гбайт; поле не должно превышать 255 символов; число индексных полей — не боле 32; длина строки — не более 2 кбайт; число строк — 10^6; открытых таблиц одновременно — не более 255.

СУБД позволяет поддерживать ссылочную целостность, задавать ограничения на поля (в том числе — на начальные значения, по умолчанию). В качестве языков программирования возможно использовать SQL, Visual Basic for Application (VBA).

В СУБД Access имеются следующие основные типы данных: текстовый, числовой, счетчик, логический, денежный, дата/время.

Основными объектами СУБД являются таблицы, формы (входные объекты), запросы, отчеты (выходные объекты). При использовании программирования впрямую дополнительно применяют такие объекты, как макросы и модули (написанные на языке VBA).

СУБД Access позволяет работать в локальном и сетевом режимах. В данной главе использован локальный режим работы СУБД Access.

Логическая модель БД. Для локального варианта выявлены, таким образом, все поля. Из изложенного видно, что частота обновления значительно ниже частоты считывания информации, т. е. имеет место статическая БД. Для нее необходимо проводить нормализацию. Результаты нормализации показаны позднее в виде схемы данных.

Реализация централизованной БД

В БД имеются четыре основных вида объектов: таблицы и формы (для ввода данных), запросы и отчеты (для вывода данных) (рис. 14.4). В локальном варианте все четыре объекта расположены в одном файле.

Рис. 14.4. Работа с базой данных

Запросы включают в себя стандартные потребности пользователя. При необходимости они легко корректируются или формируются новые. Запросы чаще всего являются промежуточными элементами между сведениями в БД и отчетами и специально не рассматриваются.

Отчеты являются выходными документами компьютера.

Таблицы и формы защищены и формируются администратором базы данных. Для пользователей они недоступны или доступны в режиме «только для чтения». В то же время запросы и отчеты полностью доступны пользователям.

Построение собственно БД. Опишем вариант технологии построения таблиц в рамках СУБД Access.

После открытия СУБД Access и выбора в промежуточном окне радиокнопки *Новая база данных* открывается окно Файл новой базы данных. В нем необходимо задать имя БД и ее адрес. После нажатия кнопки *Создать* открывается основное окно СУБД Access (рис. 14.5), в котором следует выбрать закладку *Таблицы*.

Построение структуры. Структуру таблицы проще всего сформировать с помощью конструктора. Для этого следует нажать кнопку *Создание таблицы в режиме конструктора*, при этом на экране появится основное окно для создания таблицы (рис. 14.6). Перечень полей таблиц БД «Учебный процесс» приведен в Приложении 1.

В верхней части основного окна (таблиц) указываются имена полей и соответствующие типы данных. Последние выбираются из

Рис. 14.5. Основное окно СУБД Access

поля со списком. В поле *Описание окна* возможно дать краткое содержательное описание полей таблицы.

В нижней части основного окна могут быть установлены свойства таблицы (параметры типов данных, начальные значения, условия на значения), введены индексные поля, указано условие обязательности заполнения поля таблицы.

Первичные ключевые поля, которые желательно располагать первыми в структуре, помечаются (с использованием клавиши Ctrl, если полей несколько), а затем на панели инструментов следует нажать значок с изображением ключа. Символы ключа появляются в соответствующих строках описания полей верхней части основного окна.

После завершения формирования структуры таблицы основное окно закрывают (значок «х» в верхнем правом углу). Компьютер в дополнительном всплывающем окне просит ввести имя таблицы (предлагая по умолчанию имя ТАБЛИЦАi, где i — порядковый номер) или отказаться от сохранения результата.

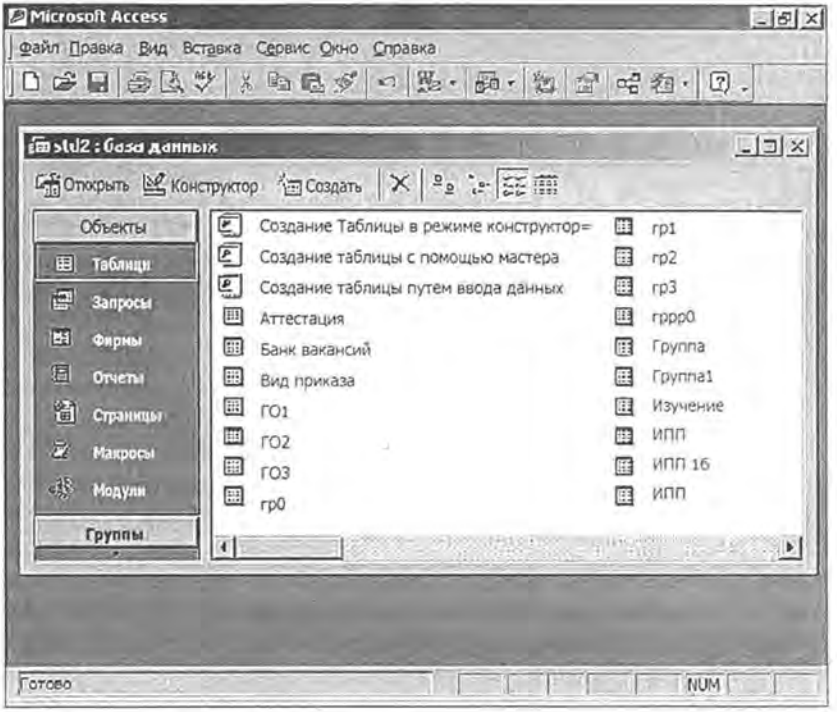


Рис. 14.6. Основное окно для создания таблиц

После введения имени таблицы (например, «Группа») и нажатия кнопки *Да* дополнительного всплывающего окна это окно закрывается и имя таблицы появляется в списке таблиц закладки *Таблицы* основного окна СУБД Access.

Описанным способом формируется структура всех таблиц.

Установление связей. Для создания схемы связей следует нажать кнопку *Схема данных* на панели инструментов Access (в основном окне на закладке *Таблицы* — рис. 14.5), при этом на экране появляется основное окно, показанное на рис. 14.7.

Во всплывающем дополнительном окне выбирают необходимые таблицы и добавляют в основное окно, после чего дополнительное окно закрывают.

Далее устанавливают связи от родительского ключа к внешнему ключу (например, «Группа.Номер группы», «Студент.Номер группы»). Для этого выделяют родительский ключ, нажимают левую кнопку мыши и, не отпуская ее, перетаскивают курсор к внешнему ключу. Появляется новое дополнительное окно (рис. 14.8), в кото-

Рис. 14.7. Дополнительное окно схемы данных

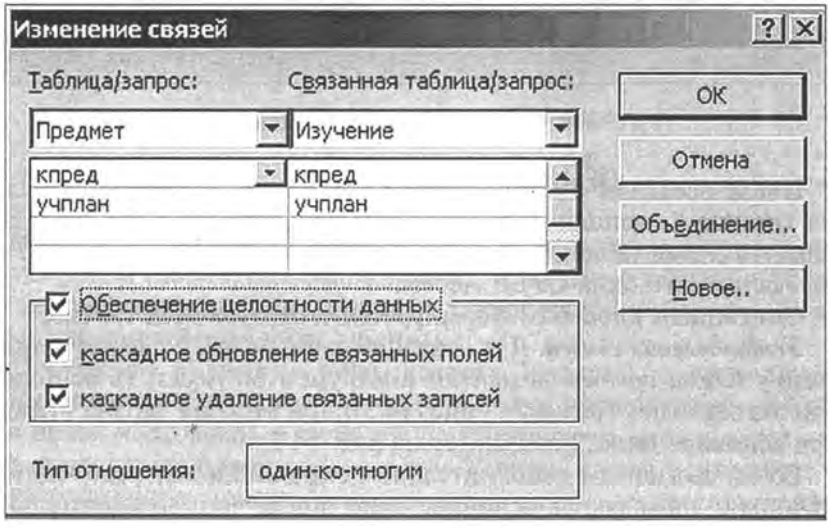


Рис. 14.8. Новое дополнительное окно схемы данных

ром указаны поля связи. Чтобы обеспечить ссылочную целостность, щелкают мышью на флажки *Обеспечение целостности данных, Каскадное обновление, Каскадное удаление.*

После закрытия дополнительного окна в основном окне схемы данных появляется соответствующая связь.

Аналогично устанавливаются и другие связи. В итоге получается схема, показанная на рис. 14.9 и характеризующая результаты нормализации.

Заполнение. Изначально (рис. 14.9) должны быть заполнены таблицы «Кафедра», «Предмет», «Предприятие». Во вторую очередь заполняют таблицы «Лаборатория», «Специальность», «Преподаватель», «Банк вакансий». После этого может быть заполнена таблица «Группа». Далее следует перейти к таблицам «Студент» и «Изучение». Затем наступает очередь таблиц «Успеваемость», «ИПП», «Оплата», «Общежитие», «Приказ». Наконец, последней заполняется таблица «Вид приказа» и «Аттестация».

Заполнение возможно проводить следующими способами:

- непосредственным диалоговым вводом данных в таблицу;
- через объект «форма»;
- из других баз данных;
- из других таблиц данной БД с помощью вычислений и запросов.

Первые два способа предпочтительны при введении небольшого объема данных. Таблицы лучше использовать при заполнении с последующим просмотром данных.

Более удобно диалоговое введение данных через формы, перечень которых приведен в ТЗ.

Покажем формирование формы «Группа» на основе одноименной таблицы.

Необходимо в основном окне СУБД перейти на закладку *Формы* и нажать кнопку *Создание формы с помощью мастера*. В открывшемся первом окне *Мастера* выбрать таблицу и ее поля, которые следует поместить в форму. В последующих окнах выбрать внешний вид, стиль формы. В последнем окне *Мастера* задать имя формы (по умолчанию или после корректировки). Нажатие кнопки *Готово* завершает создание формы, имя которой появляется в списке основного окна СУБД (закладка *Формы*).

Корректировка формы возможна нажатием кнопки *Конструктор* для выбранной формы и передвижением с помощью мыши соответствующих элементов управления. Отметим, что создание главной кнопочной формы мало отличается от описанной процедуры.

Наряду с главными формами имеются подчиненные формы, не имеющие самостоятельного значения.

Дадим краткую характеристику основных форм.

В основных формах выделим простые формы, связанные с заполнением только одной таблицы, и составные формы, позволяющие заполнять сразу две и даже три таблицы.

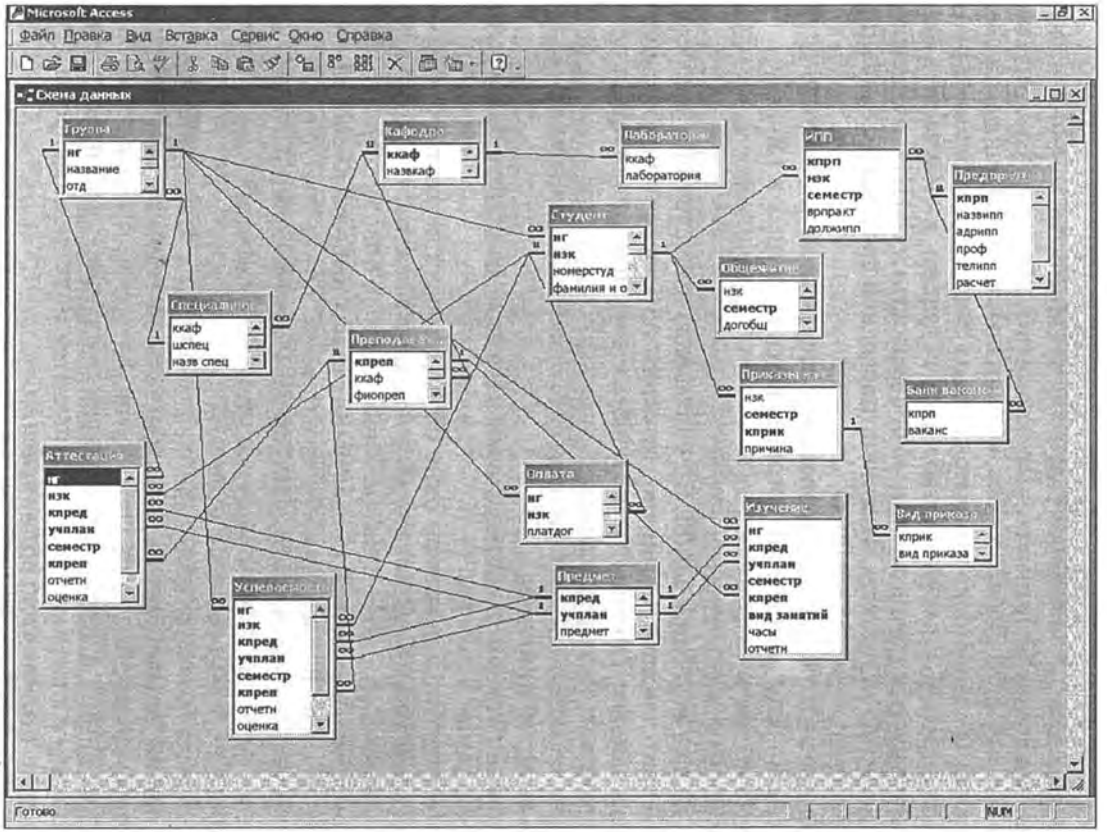


Рис. 14.9. Схема связей БД «Учебный процесс»

К простым формам относятся «Лаборатория», «Оплата», «ИПП», «Общежитие», «Студент».

Формы «ИПП» и «Общежитие» требуют введения параметра. Форма «Студент» сделана с закладками. На первой закладке *Паспорт* выделены наиболее часто используемые и наиболее важные поля. Остальные закладки (*Родственники*, *Образование*, *Личное фото*) включают поля, используемые значительно реже.

К составным формам относятся «Группа» (позволяющая заполнять данные о группе и специальности), «Изучение» (изучение и успеваемость), «Кафедра» (кафедра, преподаватель, лаборатория), «Предмет» (предмет, изучение, успеваемость), «Предприятие» (предприятие, ИПП, банк вакансий), «Приказы на студента» (приказы и вид приказов).

В ряде форм введение кода (группы, студента и т.д.) упрощается введением поля подстановки, в котором кроме кода указывается значение соответствующего понятия. Подстановка осуществляется после нажатия «треугольника» в соответствующем поле и записи формы.

При введении данных из других БД чаще всего приходится их преобразовывать. Для этого используют такую последовательность.

Необходимые таблицы из других БД импортируют в текущую таблицу. Импорт относится и к Excel- и Word-таблицам. Затем из импортированных таблиц составляют необходимые запросы-выборки с характеристиками полей таблицы, в которую необходимо ввести данные. Далее запрос-выборку переводят в запрос-обновление или запрос-добавление, реже в запрос-объединение.

Например, таблица «Студент» БД «Учебный процесс» заполняется с помощью запроса, составленного из таблицы «Студент1» другой БД.

Такой же прием применяется при использовании собственных таблиц БД «Учебный процесс». Примером использования собственных данных являются запросы, позволяющие провести вычисления полей в таблице «Группа» и заполнить таблицу «Изучение».

С технической точки зрения возможно выделить начальное и рабочее заполнение.

Начальное заполнение осуществляется в процессе создания БД, отличается большой трудоемкостью и требует автоматизации процедуры. Автоматизация осуществляется системой промежуточных запросов и таблиц и может осуществляться в диалоговом или автоматическом режиме. Первый режим используется в процессе отладки БД. Во втором случае управление системой осуществляется макросами или программами на языке Visual Basic for Applications (VBA)

при возникновении или изменении данных в определенных таблицах. Начальное заполнение может осуществляться в две стадии: частичное (для проверки работоспособности БД) и полное заполнение.

Рабочее заполнение характеризуется значительно меньшей размерностью процедуры *обновления (добавления, удаления, изменения)* данных. В связи с этим здесь более широко применим ручной *диалоговый* режим заполнения через формы или при непосредственном использовании таблиц. В то же время таблицы большой размерности, чаще всего «привязанные» к периодам времени (например, семестр), таблицы с вычисляемыми полями следует по-прежнему заполнять в автоматизированном режиме. Сюда же примыкают и процедуры архивации и автоматического изменения данных (названия групп, курса).

Осветим вопросы заполнения детальнее.

Начальное заполнение. Ряд таблиц («Группа», «Лаборатория», «Кафедра», «Преподаватель», «Специальность») имеют небольшую размерность. Они заполняются вручную в режиме диалога. Данные вводятся или непосредственно в таблицы, или через формы.

Вместе с тем в таблице «Группа» имеются вычисляемые поля, для введения данных в которые следует использовать систему запросов и таблиц как при начальном, так и при рабочем заполнении.

Более высокоразмерными являются таблицы «Предмет» и «Изучение». В силу отсутствия «электронного» источника данных информация вводится в них вручную из документов «Учебный план» и «Учебный график» соответственно. В таблицу «Изучение» вводятся также данные из таблиц «Преподаватель» и «Группа».

Рабочее заполнение. Здесь следует разделить незначительные корректировки (от одной до нескольких записей) данных и введение крупных объемов данных. Целесообразно говорить лишь о втором виде заполнения, поскольку первый («ручной») вид не представляет сложности.

Речь идет о таблицах «Студент», «Предмет», «Предприятие». Содержание «производных» таблиц («ИПП», «Группа», «Успеваемость», «Общежитие», «Оплата») следует менять через макросы в автоматическом режиме.

Значительные изменения следует ожидать в таблицах «Студент», «Предмет», «Изучение», «Успеваемость». Если для таблицы «Студент» процедура пополнения легко автоматизируется, то для таблиц «Предмет» и «Изучение» актуальным вопросом является автоматизация процедуры рабочего заполнения (фактически — автоматизация составления учебного плана и учебного графика).

Ручным остается и заполнение таблицы «Предприятие», а также введение оценок по зачетам и экзаменам. С охватом базой данных всех кафедр оценки могут вноситься в таблицу БД «Учебный процесс» непосредственно преподавателями или секретарями кафедр в диалоговом режиме.

Интерфейс пользователя. Пользователями являются не только деканат, но и договорной отдел, учебный отдел, отдел кадров. В то же время администратор базы данных (АБД) находится в деканате. Следовательно, интерфейс должен быть удобен для одного пользователя — АБД.

Интерфейс — средство взаимодействия пользователя и компьютера и включает в общем виде набор меню (кнопочных форм) и элементов управления.

В базе данных «Учебный процесс» используется кнопочная форма, более традиционная для программных продуктов фирмы Microsoft. В главной кнопочной форме выделяются три позиции (кнопки): *Заполнение*; *Использование*; *Выход из приложения*. Дополнительно вводится кнопка *Изменения* кнопочной формы. Позиция *Заполнение* защищена от несанкционированного допуска, чтобы не вносить анархию в процесс введения данных.

При нажатии кнопки *Заполнение* появляются кнопки *Деканат*, *Кафедры*, *Договорной отдел*, *Отдел кадров*, *Учебный отдел*. Все кнопки, кроме первой, «привязаны» непосредственно к формам базы данных «Учебный процесс». Имеется кнопка возврата в главную кнопочную форму. Кнопка *Деканат* снова «раскрывается» в кнопки *Студент*, *Приказы на студента*, *ИПП*, *Группа*, каждая из которых связана непосредственно с формами БД «Учебный процесс». Имеется и возврат в кнопку *Заполнение*.

Для кнопки *Использование*, в настоящее время насчитывается около 30 отчетов-документов БД «Учебный процесс».

При открытии кнопки (формы) *Использование* появляются кнопки *Деканат*, *Кафедры*, *Договорной отдел*, *Учебный отдел*, *Отдел кадров*. В форме *Деканат* имеются формы *Ведомости*, *Группа*, *Производственная практика*, *Кураторы и старосты*, *Сессия*, *Приказы на студентов*. В форме *Группа* содержатся формы *Характеристика группы*, *Список группы* и *Список группы со старостами*. В форме *Производственная практика* отражен отчет о прохождении практики. В форме *Кураторы и группы* — отчеты со списком кураторов и списком старост. В форме *Сессия* отражены отчеты о задолжниках, списки студентов, сдавших сессию с одной и двумя четверками, средний балл группы и студента, результаты сессии. В форме *Приказы на студента* имеются отчеты приказов о переводе на следующий курс, об отчисле-

нии и форма приказов о стипендии. В последней имеются приказы о стипендии отличникам и студентам, сдавшими сессию с одной и двумя четверками. В каждой форме кнопочной формы имеются формы возврата в предыдущую форму.

Алгоритм преобразования (приложения). В данном случае он составляется с использованием системы запросов, отчетов, программ с применением макросов и языка программирования VBA. Запросы и программы играют роль посредников между исходными таблицами и конечными отчетами.

Приведем примеры использования запросов и программ для формирования отчетов.

Пример 14.1. Отчет «Группа» формируется на основе запроса *Группа*. Запрос в рамках Access строится достаточно просто с использованием визуального языка QBE.

Необходимо перейти на закладку *Запросы* основного окна СУБД Access (рис. 14.10) и нажать кнопку *Создание запроса в режиме конструктора*. Во всплывающем окне следует указать имена необходимых таблиц и запросов, на основе которых строится формируемый запрос (в данном случае — на основе таблицы «Группа») и закрыть это окно. Открывается основное окно создания запроса (с использованием языка программирования QBE), в верхней части которого имеют место данные о выбранных таблицах и запросах.

В нижнюю часть основного окна последовательно вводят имена полей, используемых в запросе. Для этого в верхней части окна выделяют соответствующее поле, на которое щелкают дважды. В нижней части окна могут быть введены условия отбора данных, параметры запроса, условия агрегации данных.

После завершения всех манипуляций основное окно (формирования запроса) закрывают (значок «х» в правом верхнем углу окна). Компьютер просит сохранить запрос заданием его имени (по умолчанию ЗАПРОСi, где i — порядковый номер) либо отказаться от полученных результатов. После возможной корректировки имени и нажатия кнопки *Да* формирование запроса заканчивается и его имя появляется в основном окне СУБД (закладка *Запросы*).

Выполнение отчетов удобно осуществлять с помощью мастера. Перейдем на закладку *Отчеты* (рис. 14.11) и выберем элемент *Создание отчета с помощью мастера*. Далее работа ведется почти аналогично созданию формы в полуавтоматическом режиме. В первом окне *Мастера* выбирают таблицы (запросы) и поля (в данном примере — таблицу «Группа со всеми полями»), используемые в отчете. Во втором и третьем окнах *Мастера* выбирают поля сортировки и

Рис. 14.10. Основное окно запроса

группировки (возможно с подведением итогов). В последующих окнах *Мастера* — макет отчета и стиль оформления. В последнем окне задают имя отчета (по умолчанию — ОТЧЕТi) и нажатием кнопки *Готово* завершают построение отчета. Сформированный отчет появляется на экране, а его имя фиксируется в основном окне СУБД на закладке *Отчеты*.

Если полученный вариант отчета требует корректировки, его закрывают и на закладке *Отчеты* основного окна СУБД нажимают кнопку *Конструктор*. Далее передвигают необходимые элементы управления в отчете.

Следует отметить, что возможности языка программирования QBE ограничены. Для получения сложных запросов и отчетов приходится использовать язык программирования VBA.

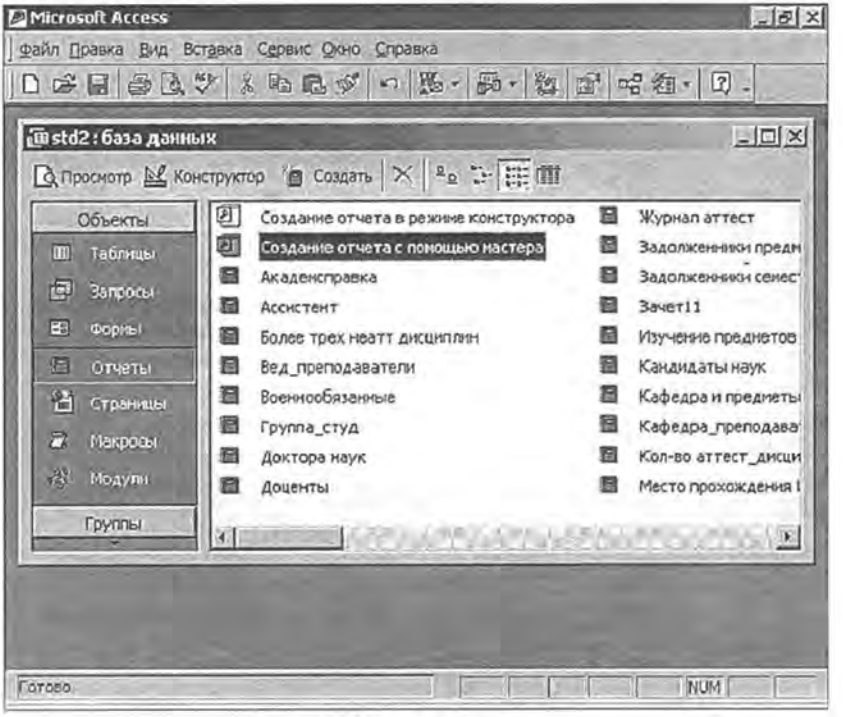


Рис. 14.11. Окно отчета

Пример 14.2. Покажем процедуру программирования. Отметим, что в VBA программные модули могут быть автономными (их можно использовать с любым объектом) или «привязанными» к форме или отчету (модуль формы, модуль отчета).

Рассмотрим программу второго типа.

Перейдем в основном окне СУБД к закладке *Формы*, выберем форму и нажмем кнопку *Конструктор*.

С помощью правой клавиши мыши выберем в контекстном меню для формы в целом или отдельного поля элемент *Свойства*. В открывшемся всплывающем окне перейдем на закладку *События* и в строке выбранного события нажмем значок троеточия. В появившемся окне *Построитель* выберем элемент *Программы* и нажмем кнопку *OK*.

Программа может иметь вид

```
Option Compare Database
Private Sub Кнопка12_GotFocus()
```

```
Кнопка8.Enabled = False
End Sub
```

При этом две первые и последняя строки программы «набраны» компьютером. Имя модуля формируется автоматически и запоминается при закрытии формы.

Аналогично формируются модули для отчетов.

Автономные модули строятся в основном окне СУБД на вкладке *Модули*.

Защита осуществляется паролями и режимом «только для чтения» для базы данных в целом и/или для отдельных таблиц. В Access доступ может формироваться с помощью объекта *Страницы*, устанавливающего, какой уровень доступа имеют определенные поля таблицы (путем формирования таблицы доступа). Сбои оборудования компенсируются работой СУБД (кратковременные сбои) и автоматическим формированием через определенный промежуток времени резервной копии (долговременные сбои). Ручное выполнение резервной копии следует проводить и при всех процедурах обновления, в которых возможно искажение или потеря данных.

14.4. РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

Проектирование распределенной БД

Описанная централизованная БД «Учебный процесс» была реализована на СУБД Access, апробирована и показала хорошую работоспособность. В то же время в процессе апробации выявились дополнительные требования конечного пользователя, которые в итоге привели к построению сетевого варианта (распределенной) БД.

Для распределенной БД в значительной мере имеют место этапы построения, характерные для локального централизованного варианта. Поскольку они были подробно рассмотрены, ограничимся изложением отличий сетевого варианта от локального.

Анализ требований. Апробация централизованной БД показала, что для нее характерны централизованное заполнение и — что более неудобно — централизованное использование: в каждый промежуток времени может работать только один пользователь. Это приводит к необходимости разделения по времени работы различных подразделений института.

Следует отметить значительные изменения «ручной» структуры БД (рис. 14.12). Появляются новые структурные элементы: второй

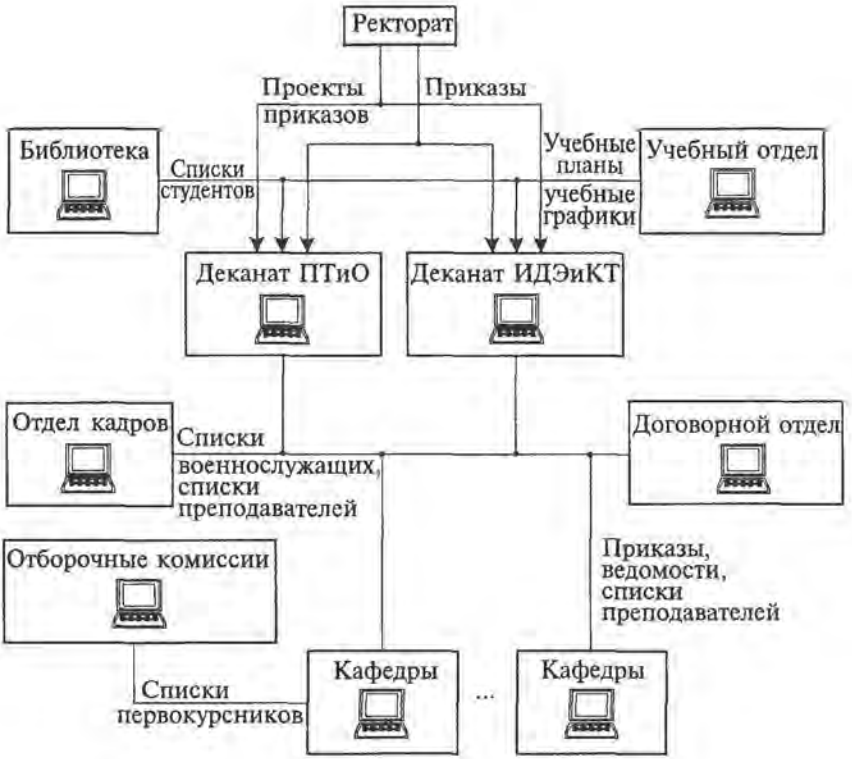


Рис. 14.12. Изменения структуры БД

деканат, кафедры. Структура связей таблиц не претерпевает существенных изменений, однако сильно меняется распределение данных между подразделениями в соответствии с рис. 14.13. Появляются и новые поля в таблицах.

По сравнению с локальным вариантом появилась необходимость в дополнениях.

Таблицы:

Выпускник.

Список научных трудов преподавателей.

Индивидуальный план преподавателя в данном учебном году (госбюджет).

Индивидуальный план преподавателя в данном учебном году (внебюджетная форма).

Перечень нормативной документации.

Оборудование.

Добавить поля в таблицы:

Выпускник.

Кроме полей таблицы «Студент» и «Оплата» добавить поля

Дата окончания института

Приказ об отчислении в связи с окончанием

N диплома

Специальность

Квалификация.

Преподаватель

Количество кандидатов наук.

Количество докторов наук.

Количество доцентов.

Количество ст. преподавателей.

Количество ассистентов.

Количество (общее) преподавателей.

Год рождения.

Возраст.

Стаж работы в высшей школе.

Стаж работы в институте.

Год поступления в институт.

Научно-педагогический стаж

Год окончания института

Специальность по образованию.

Дата последнего избрания.

Дата последнего повышения квалификации.

В таблице «Успеваемость» учесть в качестве отчетности аттестацию, госэкзамен, защиту дипломного проекта (с оценками).

Отчеты

Приложение к диплому.

Академическая справка.

В СУБД Access оказалась недостаточна степень защиты от сбоев в работе БД от различного рода ошибок, вызванных как проблемами с компьютером, так и ошибками пользователя.

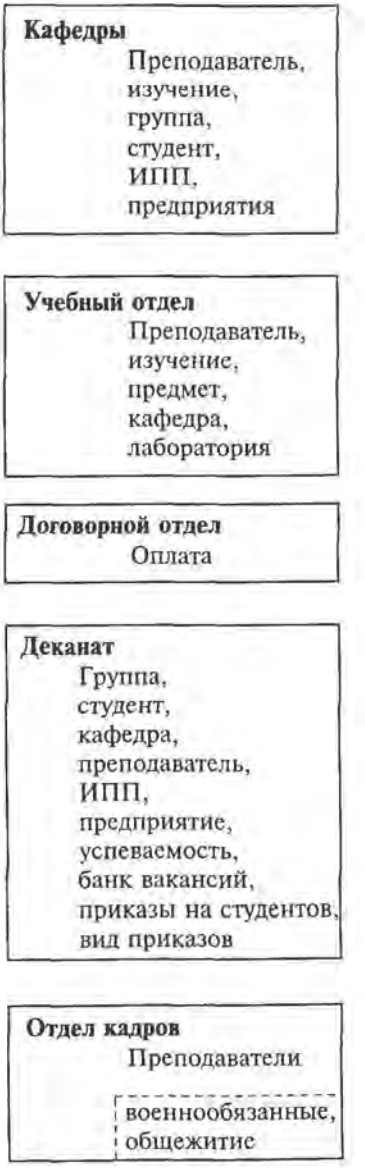


Рис. 14.13. Распределение данных таблиц БД между подразделениями

В Access отсутствуют системы автоматического восстановления БД, что приводит к необходимости ежедневного создания резервной копии БД.

Таким образом, возникла необходимость в создании распределенной информационной системы для учебного процесса, позволяющей более эффективно взаимодействовать различным подразделениям.

Техническое задание

Создаваемая распределенная база данных (РБД) должна удовлетворять следующим требованиям.

- Интерфейс РБД должен быть рассчитан на пользователя начального уровня.
- РБД должна быть построена на основе хорошо себя зарекомендовавшей, широко распространенной РСУБД.
- Работа с РБД не должна требовать от пользователя каких-либо познаний в области языков высокого уровня.
- В связи с постоянным ростом объема информации, обрабатываемой в РБД, необходимо предусмотреть возможность ее масштабируемости (возможность подключения дополнительных файлов РБД при превышении допустимого количества записей, хранимых в основной РБД).
- Распределенная СУБД (РСУБД) должна иметь встроенные средства защиты от сбоев в работе РБД.
- РСУБД должна поддерживать работу с хранимыми процедурами.
- РСУБД не должна иметь существенных ограничений на обрабатываемый ею объем информации и должна быть достаточно быстродействующей, чтобы обеспечить комфортную работу пользователя.
- РСУБД должна быть нетребовательна к платформе, на которую устанавливается и должна быть проста в установке и конфигурировании.

Концептуальная модель меняется мало. Предложено расширить таблицу «Преподаватель» такими полями, как *Год рождения*, *Дата поступления на работу*, *Стаж научно-педагогической деятельности*, *Перечень читаемых дисциплин*, *Базовое образование*, *Дата увольнения*; ввести в БД «Учебный процесс» две новых таблицы: «Труды сотрудников (преподавателей)», напоминающую по своему виду список трудов научного работника, «Выполнение нагрузки преподавателем». Эти таблицы несущественно усложняют структуру БД «Учебный процесс».

Выбор СУБД. В силу названных ранее обстоятельств по-прежнему используется реляционная модель данных. Уверенно удовлетво-

рить перечисленным требованиям ТЗ в рамках СУБД Access малоперспективно. В силу файл-серверного режима Access количество клиентов ограничено четырьмя, тогда как ТЗ определяет уже пять пользователей. Их число в дальнейшем может увеличиться.

Следует обратиться к СУБД, изначально созданной для работы в сетевом режиме. Из них отберем наиболее широко распространенные в России InterBase, SQL Server, Sybase.

Сравнительные характеристики названных СУБД по принятым в соответствии с ТЗ критериям приведены в табл. 14.1.

Таблица 14.1

Выбор СУБД

Характеристики	InterBase	SQL Server	Sybase
Механизм блокировки	Многоверсионность (изменение столбцов)	Страница данных	Страница данных
Хранение	Обрезание концов типов данных	255 символов	255 символов
Производительность	Многоверсионность	Влияние на блоки	Влияние на блоки
Установка (инсталляция)	Автоматическое распределение пространства	Сложный для пользователя процесс	Сложный для пользователя процесс
Обслуживание	Архивация в любое время	Архивация по ночам	Архивация по ночам
Конфигурация и настройка	Автоматическая конфигурация	Много конфигурационных опций	Много конфигурационных опций
Восстановление при сбоях	Автоматическое восстановление	Сложность восстановления	Сложность восстановления
Требования к ресурсам	Ядро — 2 Мбайт, на диске — 8 Мбайт	600 Мбайт для установки, 24 Мбайт оперативной памяти	10 Мбайт на диске
Наличие широко применяемого языка программирования	SQL, Object Pascal	Transact-SQL, VBA	—
Структура	InterBase + Delphi	Access + ODBC + SQL Server	—

Из нее следует, что наилучшим образом техническому заданию удовлетворяет СУБД InterBase, обеспечивающая одновременный доступ к данным, имеющая мощную систему авторизации и совершенную систему защиты от сбоев.

Использование InterBase в среде Delphi только увеличивает преимущества СУБД.

После выбора СУБД могут быть уточнены требования к системе компьютеров.

Минимальная конфигурация сервера

- Intel Pentium 166 МГц.
- HDD 1 Гбайт.
- ОЗУ 24 Мбайт.
- Сетевая карта Ethernet 10 Мбайт (рекомендуется 100 Мбайт).

Клиентская машина

- Intel Pentium 100 МГц.
- HDD 1 Гбайт.
- ОЗУ 16 Мбайт.
- Сетевая карта Ethernet 10 Мбайт.

Программный комплекс может функционировать как в сети с выделенным сервером, так и в одноранговой сети. Операционной системой сервера может быть: Windows NT, Novell NetWare 4.X, 5.X, Windows-95, Windows-98.

При использовании в качестве серверного программного обеспечения Novell NetWare доступ к РБД с машин клиента необходимо реализовать по протоколу Novell IPX/SPX, при использовании WindowsNT — по протоколу NetBeui, при использовании одноранговой сети на основе Windows-9X доступ должен осуществляться по протоколу TCP/IP.

На сервере должны быть установлены InterBase 4.2 (или выше), BDE 5.01 (или выше)

На компьютере пользователя должна стоять операционная система Windows 9X или Windows NT, клиентская часть InterBase 4.2 (или выше) и BDE 5.01 (или выше).

Зашита данных осуществляется с помощью системы паролей, разграничивающих доступ разных пользователей к различным таблицам. Настройка паролей осуществляется с помощью утилиты Server Manager, поставляемой с РСУБД InterBase. Защита РБД осуществляется и путем создания резервных копий. Резервное копирование РБД осуществляется с помощью утилиты Server Manager.

В РСУБД InterBase существует и способ защиты РБД под названием «система зеркалирования (Shadow)».

Фрагментация и локализация. Обычно предполагается горизонтальная и вертикальная фрагментация. Горизонтальная фрагментация предусматривает деление однотипных данных по узлам. В рассматриваемом случае она означает разделение задач по клиентам и условное деление данных между ними с помощью приложений, размещаемых на компьютерах-клиентах. Локализация (привязка данных к узлам) в клиентском режиме также предельно упрощена, поскольку данные размещены на сервере.

Реализация распределенной БД

При использовании режима клиент—сервер возникают такие задачи для сервера и клиентов.

Для сервера

1. Построение структуры системы таблиц на сервере, возможно с системой запросов (видов) для заполнения таблиц, с установлением связей между таблицами.
2. Заполнение таблиц данными.
3. Обеспечение доступа через виды, запросы и хранимые процедуры.

Для клиентов

1. Разработать необходимые приложения.
2. Построить соответствующие интерфейсы пользователей.

В данной главе рассмотрим локальный вариант режима клиент—сервер. Особенности формирования удаленного варианта с одно- и многоуровневой структурой обсудим в гл. 15.

Собственно база данных. Необходимо выполнить следующие работы.

- Построение таблиц.
- Установление связей между таблицами для осуществления ссылочной целостности в РБД.
- Использование хранимых процедур для обеспечения ссылочной целостности.
- Создание системы генераторов для обеспечения автоматического заполнения полей, требующих автонумерации.
- Загрузка таблиц данными из существующей БД «Учебный процесс».

Построение структуры таблиц. Все операции по созданию структуры таблиц РБД в InterBase осуществляются при помощи утилиты Windows ISQL (WISQL), поставляемой в комплекте с сервером РБД (рис. 14.14).

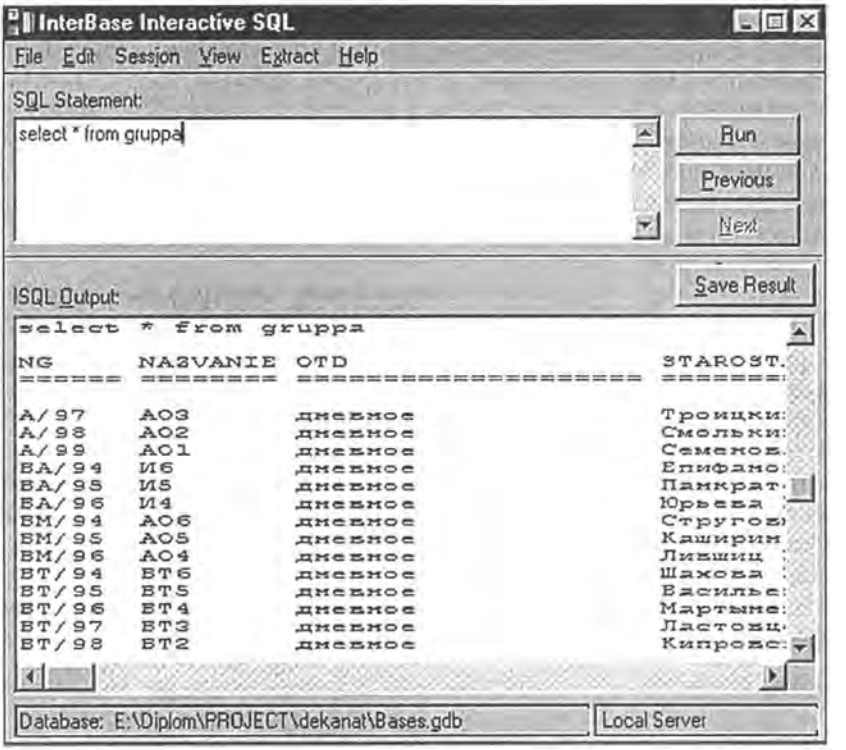


Рис. 14.14. Окно Windows ISQL

Создание таблиц и других объектов РБД выполняется на языке SQL.

Язык QBE серьезно ограничивает разработчика и позволяет ему создавать запросы только по готовым шаблонам, которые были заложены создателями языка.

Приведем пример создания таблицы gruppa:

```
/* Table: GRUPPA, Owner: SYSDBA */
CREATE TABLE GRUPPA (NG VARCHAR(6) CHARACTER SET
WIN1251 NOT NULL,
    NAZVANIE VARCHAR(6) CHARACTER SET WIN1251,
    OTD VARCHAR(20) CHARACTER SET WIN1251,
    STAROSTA VARCHAR(20) CHARACTER SET WIN1251,
    STAR_POLN VARCHAR(50) CHARACTER SET WIN1251,
    KURATOR VARCHAR(20) CHARACTER SET WIN1251,
    KURS VARCHAR(50) CHARACTER SET WIN1251,
```

```
    SH_SPEC INTEGER NOT NULL,
    GOD_NABOR INTEGER,
    STUDENTOV INTEGER,
    DEVUSHEK INTEGER,
    UNOSHEY INTEGER,
    BESPLATN INTEGER,
    CHAST INTEGER,
    POLNPLATN INTEGER,
CONSTRAINT GRUPPRIMKEY1 PRIMARY KEY (NG));
```

Последняя строка указывает на то, что поле NG (номер группы) является первичным ключом. Инструкция типа SH_SPEC INTEGER NOT NULL указывает на то, что необходимо создать колонку sh_spec типа Integer с обязательным ненулевым значением.

Всего РБД насчитывает 18 таблиц:

- ArhivStud — архивная таблица, содержащая информацию об успеваемости студентов, за прошедшие семестры;
- Gruppa — список всех групп;
- IPP — таблица, хранящая информацию о прохождении студентами производственной практики;
- Izuchenie — таблица с информацией об изучаемых предметах в группах;
- Kafedra — список кафедр;
- Laba — список;
- Obshegit — список студентов, живущих в общежитиях;
- Oplata — информация об оплате студентами своего обучения;
- Predmet — список преподаваемых предметов;
- Predpr — список предприятий, на которых проходят производственную практику студенты;
- Prepod — список преподавателей;
- Prikaz — приказы;
- Specialnost — список специальностей;
- Student — список обучающихся студентов и сведения о них;
- TempUsp — промежуточная таблица, необходимая для генерации сложных отчетов, которые не могут быть организованы только с использованием запросов;
- Uspevaem — информация о сдаче сессии студентами;
- Vid_prikaz — виды приказов.

После создания таблиц программист может извлечь из РБД SQL-команды, которые он вводил в процессе создания таблиц. Такая информация, получаемая из РБД, называется метаданными. Текстовый файл, содержащий метаданные, может быть в случае необ-

ходимости повторно выполнен, что позволит в случае полной потери РБД, восстановить ее структуру без повторного, утомительного набивания всех SQL-команд. Для получения подобной информации необходимо в утилите Windows ISQL выбрать в меню Extract/SQL Metadata for DataBase, после чего указать имя файла, в котором необходимо сохранить метаданные. Листинг SQL-программ разработанной РБД приведен в Приложении 2.

Установление связей между таблицами. Для создания связей в WISQL используются инструкции вида PRIMARY KEY (NG) — создание основного ключа по полю NG; FOREIGN KEY (NG) REFERENCES GRUPPA (NG) — создание внешнего ключа NG с подключением к основному ключу NG таблицы GRUPPA.

Структура таблиц и связей разработанной РБД представлена на рис. 14.15. Связи и таблицы в РБД по возможности сохранены для осуществления переноса и конвертирования информации из ранее описанной БД.

Создание хранимых процедур. Хранимая процедура в СУБД InterBase может применяться как в алгоритме приложения, так и, наряду с триггерами, в собственно базах данных для каскадного удаления и обновления.

Клиентской части нужно передать параметры в хранимую процедуру. После окончания работы хранимая процедура передает результаты клиентскому приложению. Такой подход позволяет также снизить требования к производительности компьютера пользователя, работающего с РБД.

Приведем пример хранимой процедуры, осуществляющей удаление ключевых записей из таблицы Kafedra:

```
ALTER PROCEDURE KAFEDRA_DEL (DKKAF INTEGER)
AS
declare variable dsh_spec integer;
declare variable dng varchar(6);
declare variable dnzk varchar(10);
declare variable dkprikaz integer;
declare variable dkprep integer;
BEGIN
for select sh_spec from specialnost
where kkaf=:dkkaf
into :dsh_spec
do
begin
```

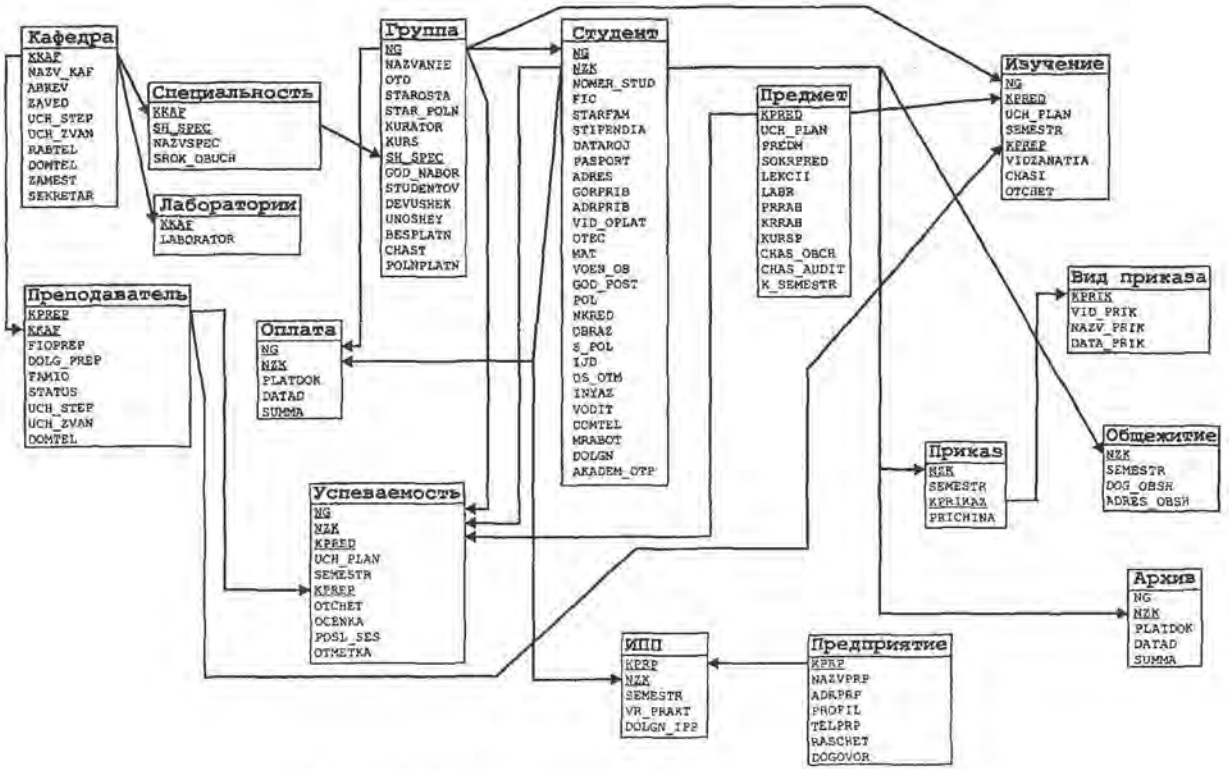


Рис. 14.15. Схема связей таблиц БД «Учебный процесс»

```
for select ng from gruppa
where gruppa.sh_spec=:dsh_spec
into :dng
do
begin
for select nzk from student
where student.ng=:dng
into :dnzk
    do
    begin
    delete from ipp where ipp.nzk=:dnzk;
    delete from uspevaem where uspevaem.nzk=:dnzk;
    delete from atestacia where atestacia.nzk=:dnzk;
    delete from oplata where oplata.nzk=:dnzk;
    delete from obshegit where obshegit.nzk=:dnzk;

        for select kprikaz from prikaz
        where prikaz.nzk=:dnzk
        into :dkprikaz
        do delete from vid_prikaz where vid_prikaz.kprik=:dkprikaz;

    delete from prikaz where prikaz.nzk=:dnzk;
    end
    delete from student where student.ng=:dng;
    delete from izuchenie where izuchenie.ng=:dng;
    delete from gruppa where gruppa.ng=:dng;
end
    delete from specialnost where sh_spec=:dsh_spec;
end

for select kprep from prepod where kkaf=:dkkaf
into :dkprep
do begin
    delete from izuchenie where kprep=:dkprep;
    delete from uspevaem where kprep=:dkprep;
    delete from atestacia where kprep=:dkprep;
    end
delete from prepod where kkaf=:dkkaf;
delete from laba where laba.kkaf=:dkkaf;
delete from kafedra where kkaf=:dkkaf;
suspend;
END
```

Данная процедура KAFEDRA_DEL в качестве параметра получает номер кафедры DKKAF, после чего сначала осуществляет удаление записей в подчиненных таблицах и лишь затем ключевой записи в таблице KAFEDRA.

Многие подчиненные таблицы сами имеют подчиненные таблицы. Таким образом, перед удалением данных в них необходимо осуществить удаление данных в подчиненных им таблицах и лишь потом — в родительских таблицах. Выйти из сложившегося положения позволяет поддерживаемая в InterBase вложенность процедур языка SQL. Благодаря этому существует возможность реализовать достаточно сложную процедуру каскадного удаления, содержащую в себе до 16 вложений при размере подобной процедуру не более 48 кбайт.

В БД «Учебный процесс» имеется несколько хранимых процедур, которые выполняют следующие работы:

- Gruppa_Del — каскадное удаление в таблице Gruppa и связанных с ней таблицах;
- Kafedra_Del — каскадное удаление в таблице Kafedra и связанных с ней таблицах;
- Predmet_Del — каскадное удаление в таблице Predmet и связанных с ней таблицах;
- Predpr_Del — каскадное удаление в таблице Predpr и связанных с ней таблицах.
- Prepod_Del — каскадное удаление в таблице Prepod и связанных с ней таблицах;
- Prikaz_Del — каскадное удаление в таблице Prikaz и связанных с ней таблицах;
- Specialnost_Del — каскадное удаление в таблице Specialnost и связанных с ней таблицах;
- Student_Del —каскадное удаление в таблице Student и связанных с ней таблицах.

Заполнение (загрузка таблиц данными из существующей БД «Учебный процесс»), как и в централизованной БД, возможно либо прямо в таблицу, либо в формы. Состав форм описан в интерфейсе пользователя.

В отличие от централизованной БД в данной БД часть данных заполняется автоматически через генераторы.

Создание системы генераторов. В явном виде InterBase не поддерживает полей типа «автонумерация». Вместо этого в InterBase существует механизм, называемый «генераторами». Генератор — некая переменная, значение которой может быть получено и увеличено на некоторое значение при помощи встроенной функции GEN_ID.

Обычно генераторы используют в триггерах, при этом текст триггера может быть следующим:

```
CREATE TRIGGER TI_CLIENTS FOR CLIENTS
ACTIVE BEFORE
INSERT POSITION 0
AS
BEGIN IF (new.CLIENT_ID IS NULL) THEN
CLIENT_ID=GEN_ID(<имя генератора>, 1);
END,
```

где 0 — начальное значение; 1 — шаг приращения в текущем значении генератора.

Механизм генераторов гарантирует, что даже при конкурентном (параллельном) вызове функции GEN_ID каждому пользователю будет выдаваться уникальное значение. Последнее значение генератора всегда запоминается в БД, поэтому разработчику не нужно заботиться о восстановлении его максимального значения после подсоединения к БД. Генераторы являются переменными типа integer (longint).

Программный комплекс (БД) «Учебный процесс» содержит три генератора, предназначенных для генерации уникальных кодов:

- Gen_kkaf — кафедр;
- Gen_kpred — предмета;
- Gen_Kprp — предприятия.

Интерфейс пользователя. Речь пойдет об интерфейсе клиента, поскольку интерфейс сервера обычно очень прост.

Интерфейс РБД должен быть реализован способом, интуитивно понятным для пользователя-непрофессионала. От пользователя требуются лишь познания в его предметной области.

В качестве средства реализации интерфейса пользователя выбран пакет разработки программного обеспечения Delphi 5, основанный на языке высокого уровня Object Pascal. Пакет позволяет создавать приложения под графические операционные системы семейства Windows. Delphi 5 разрабатывался коллективом той же компании Borland, что и РСУБД InterBase, и потому встроенная ими в Delphi 5 поддержка InterBase является наиболее полной и оптимально функционирующей.

Для создания программ, осуществляющих взаимодействие пользователя с РБД, в системе Delphi традиционно используется механизм Borland Database Engine (BDE).

В стандартную поставку BDE входит два набора драйверов.

• Первый набор предназначен для файл-серверных СУБД dBase, Paradox, FoxPro, Access и данных в текстовом формате. Эти драйверы реально представляют собой весьма сложные программы, выполняющие множество функций СУБД.

• Второй набор ориентирован на клиент-серверные РСУБД InterBase, IBM DB2, Informix, Oracle, Sybase и Microsoft SQL Server. Этот набор называется SQL Links. Такие драйверы устроены проще. Они только передают запросы и команды из BDE в РСУБД и получают обратно результаты их выполнения. Всю работу по обработке данных выполняет РСУБД.

Фирмой Microsoft был разработан и стандартный протокол ODBC (Open Database Connectivity Interface, открытый интерфейс взаимодействия с БД), напоминающий работу BDE. Драйверы ODBC выпущены для всех без исключения СУБД, и разработчик может использовать в BDE как драйверы SQL Links, так и драйверы ODBC.

Разработка интерфейса РБД на любом языке высокого уровня является сложной и кропотливой задачей, в отличие от реализации интерфейса в Access, где разработчик руководствуется стандартными шаблонами и методами построения запросов, отчетов и форм.

Интерфейс пользователя БД «Учебный процесс» насчитывает свыше сорока форм, из которых порядка двадцати необходимы для генерации отчетов.

Далее предполагается, что разработчик знаком с работой приложения Delphi в рамках автоматизации программирования и потому процедура создания интерфейса опускается.

При использовании интерфейса возможно обращаться к системе меню или экранных форм. Напомним, что в данном параграфе рассматривается случай с одним пользователем.

Общая схема интерфейса пользователя показана на рис. 14.16. Кратко опишем его элементы *Заполнение*, *Отчеты*, *Сервис*.

Основная экранная форма для элемента *Заполнение* с системой закладок (кнопок) показана на рис. 14.16. Как видно, доступ возможен через систему меню или экранных форм.

На каждой из перечисленных закладок существует набор кнопок, позволяющих выполнять те или иные действия в программе. Для удобства пользователя все действия, вызываемые по нажатию кнопок, могут быть выполнены также с помощью системы меню.

Порядок заполнения таблиц тот же, что и в централизованной БД.

Рассмотрим одну из экранных форм. При нажатии кнопки *Специальность* появляется экранная форма, показанная на рис. 14.17.

Данные могут вводиться пользователем, как в поля ввода, так и непосредственно в таблицу, расположенную в нижней части фор-

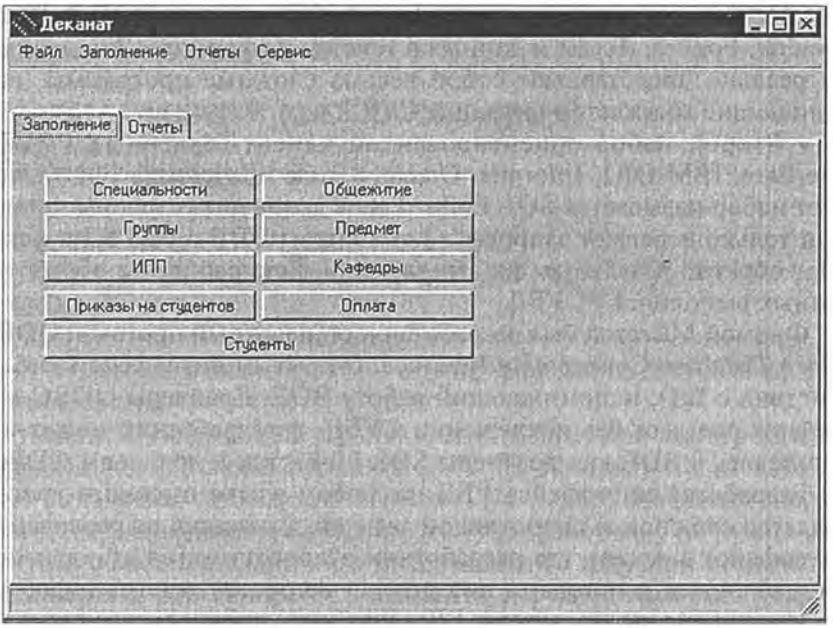


Рис. 14.16. Основная экранная форма элемента Заполнение

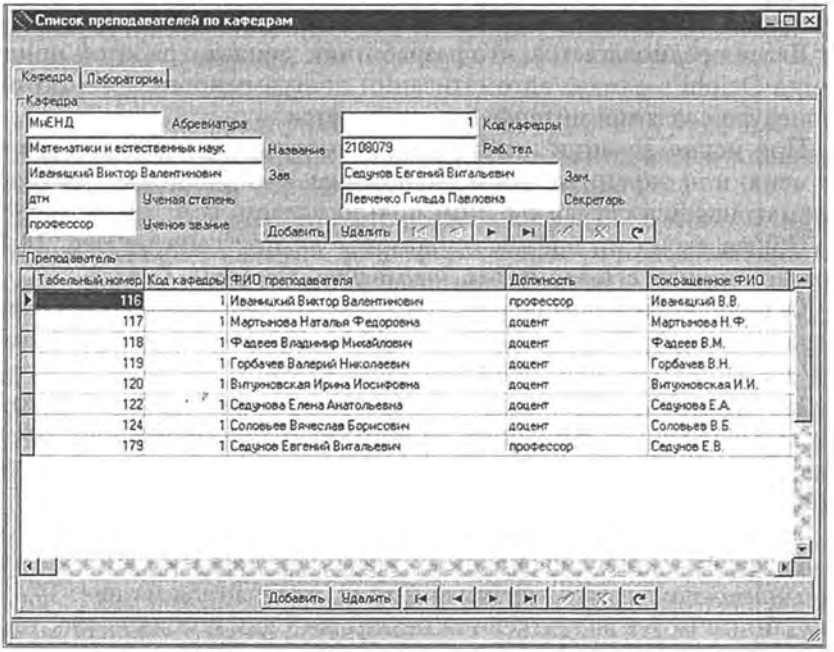


Рис. 14.17. Форма для заполнения данных о специальностях

мы. Поле *Кафедра* связано с таблицей «Кафедра», поэтому оно может принимать только те значения, которые представлены в таблице «Кафедра». Данное поле для исключения ошибок ввода пользователя представлено в виде выпадающего списка, содержащего значения из таблицы «Кафедра». Остальные поля независимые, поэтому могут содержать любые данные. Навигация по записям, а также удаление и добавление записей осуществляется кнопками навигатора *Добавить*, *Удалить*.

Замечание. Надо помнить, что при удалении записи из таблицы «Специальность» происходит каскадное удаление в связанных с ней таблицах групп студентов, обучающихся по этой специальности, данные об обучении этих студентов, их проживании в общежитии.

Закладка *Отчеты* содержит в себе закладки.

- Деканат.
- Группа.
- Сессия.
- Приказы.
- Отдел кадров.
- Договорный отдел.
- Кафедра.

Рассмотрим содержание одной из закладок.

Закладка *Деканат* содержит кнопки, генерирующие следующие отчеты.

- Ведомости — вывод на печать зачетных и экзаменационных ведомостей.
- Список старост — документ со списком старост по всем группам.
- Прохождение ИПП — документ, содержащий информацию о прохождении студентами производственной практики.
- Список кураторов групп.

В данной БД предусмотрен вариант с хранилищем данных.

После того как пользователь заполнит всю информацию о сдаче студентами сессии и сгенерирует в подсистеме ведения отчетов все необходимые и относящиеся к данной сессии документы, можно запустить процедуру автоматической очистки РБД от студентов, не сдавших сессию, а затем выполнить перевод всех студентов, успешно сдавших сессию, на следующий курс.

Процедура переноса в хранилище (архив) осуществляется через элемент *Сервис* головного меню БД с помощью экранной формы, показанной на рис. 14.18 и кнопки *Перенос данных в архив* из таблицы «Успеваемость». После переноса данные в таблице «Успеваемость» удаляются.

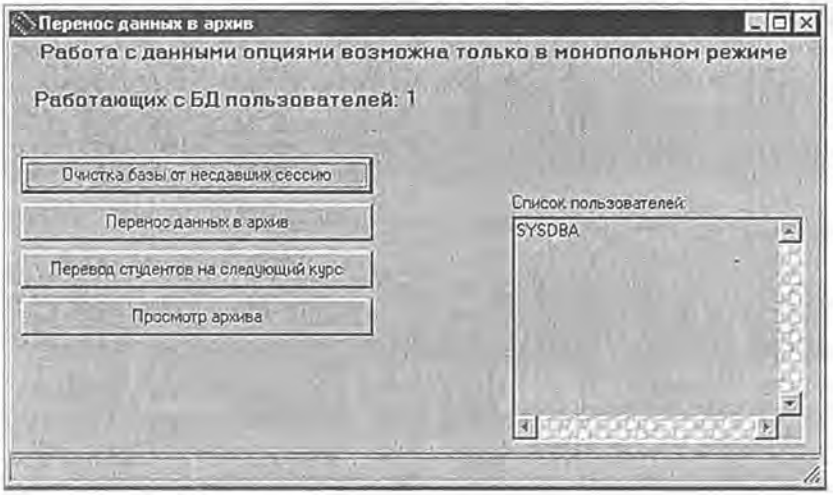


Рис. 14.18. Перенос данных в архив

В связи с большим объемом данных, перерабатываемых процедурами *Очистка базы от несдавших сессию*, *Перенос данных в архив* и *Перевод студентов на следующий курс*, работа с ними возможна только в монопольном режиме. В противном случае кнопки, запускающие эти процедуры, становятся недоступными. Программа автоматически выводит в окне *Список пользователей* имена пользователей, работающих в текущий момент с РБД.

Из рис. 14.18 видно, что в данный момент времени с РБД работает один пользователь с именем SYSDBA.

После нажатия на кнопку *Очистка базы от не сдавших сессию* программа спросит пользователя о согласии на удаление данных и после утвердительного ответа произведет удаление.

Нажатие на кнопку *Перенос данных в архив* приведет к переносу данных о сдаче текущей сессии студентами в архив, после чего будет очищена таблица «Успеваемость», находящаяся на закладке *Успеваемость* экранной формы «Предмет».

После выполнения двух вышеописанных процедур пользователь может осуществить перевод студентов на следующий курс. Для этого необходимо нажать на кнопку *Перевод студентов на следующий курс*. Данная процедура выполняет обновление данных в таблице «Группа», находящейся на одноименной закладке. В результате работы процедура увеличит номер группы на единицу, после чего осуществит коррекцию названий групп, выделив в названии группы последний символ, характеризующий номер группы, и увеличив его значение на единицу.

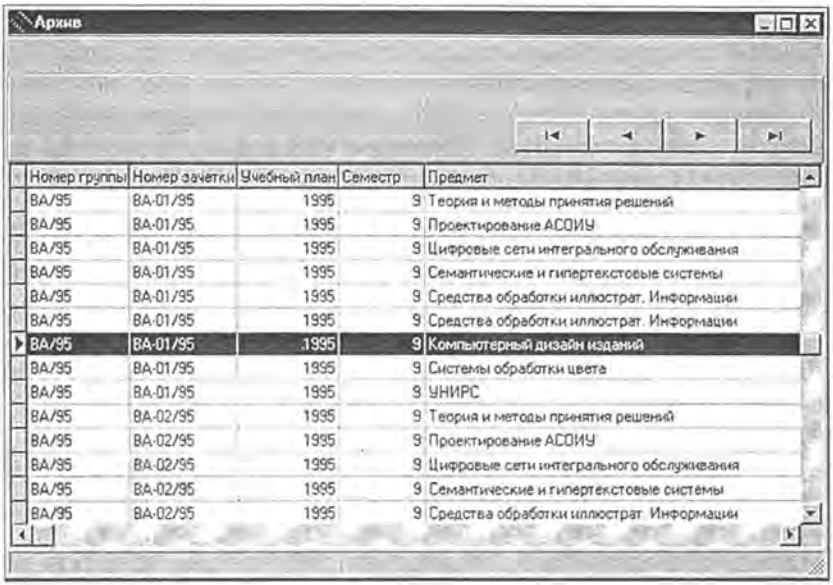


Рис. 14.19. Архив успеваемости студентов

Кнопка *Просмотр архива* открывает экранную форму изображенную на рис. 14.19, которая содержит перенесенную ранее в архив информацию об успеваемости студентов.

Данная информация доступна пользователю только для просмотра. Экранная форма содержит таблицу со следующими полями.

- Номер группы.
- Номер зачетной книжки.
- Учебный план.
- Семестр.
- Предмет.
- Преподаватель.
- Отчетность.
- Оценка.
- Отметка.

Алгоритм преобразования. В его построении участвуют:

- хранимые процедуры;
- виды серверной части;
- формы, связанные с клиентской частью.

Хранимые процедуры позволяют в значительной мере снизить объемы информации, передаваемые по сети (трафик). К ним в БД относится Grup_Sum, которая осуществляет подсчет числа «плат-

ных», «частично платных» и «бесплатных» студентов, обучающихся в каждой конкретной группе, а также осуществляет подсчет числа юношей и девушек в группе.

Создание видов (View) таблиц служит для повышения быстродействия системы запросов. Приведем команды по созданию вида EKZAMVIEW из РБД «Учебный процесс»:

```
/* View: EKZAMVIEW, Owner: SYSDBA */
CREATE VIEW EKZAMVIEW (NZK, OTCHET, SEMESTR, SH_SPEC,
POSL_SES, FIO, KURS) AS
SELECT
uspevaem.nzk,otchet,uspevaem.semestr,gruppa.sh_spec,
posl_ses,student.fio,
gruppa.kurs
from uspevaem, gruppa, student
where uspevaem.ng=gruppa.ng and
uspevaem.nzk=student.nzk and semestr <> 12
group uspevaem.nzk,otchet,uspevaem.semestr,sh_spec,posl_ses,
student.fio,gruppa.kurs by
having otchet='экз';
```

Как видно из примера, выборку данных можно осуществлять одновременно из неограниченного числа связанных таблиц. Так, данный вид осуществляет выборку записей из трех связанных таблиц, после чего группирует полученную информацию по полям nzk, otchet, semestr, sh_spec, posl_ses, fio, kurs. Этот вид используется при генерации отчета по успеваемости студентов в текущем семестре. Информация, полученная из этого вида, в дальнейшем перерабатывается интерфейсной частью и используется для генерации отчетов типа:

- студенты, не сдавшие сессию;
- студенты, сдавшие сессию на все пятерки;
- студенты, сдавшие сессию на пятерки и одну четверку;
- студенты, сдавшие сессию на пятерки и две четверки.

В программе имеются следующие виды, которые используются:

- EkzamView — для генерации отчетов о сдаче студентами сессии;
- GruppaView — для ускорения доступа к связанным полям;
- StudentView — для ускорения доступа к связанным полям;
- ZadolgView — для генерации отчетов о сдаче студентами сессии.

Программа содержит большое количество алгоритмов, необходимых для обработки тех или иных действий пользователя, генерации сложных отчетов, взаимодействия с РБД и переработки информации. Многие из этих алгоритмов было практически невозможно

реализовать, используя лишь стандартные конструкции Object Pascal, поэтому в них наряду с Pascal использовался также и язык SQL. Такой симбиоз языков позволяет в значительной мере ускорить выполнение данных алгоритмов. В качестве примера программного кода, содержащего как Pascal, так и SQL, приведем фрагмент процедуры генерирующей «зачетную (экзаменационную) ведомость».

```
procedure TForm11.Button5Click(Sender: TObject);
var
t1:string[10];
begin
t1:=datamodule2.izuchenie.fieldbyname('otchet').asstring;
```

Загрузка в переменную t1 текущего значения колонки otchet таблицы izuchenie

```
if t1='зач' then form12.QRLabel1.caption:='Зачетная ведомость';
```

Если t1=зач, то вывести в отчет 'Зачетная ведомость'

```
if t1='аттес' then form12.QRLabel1.caption:='Аттестационная ведомость';
if t1='экз' then form12.QRLabel1.caption:='Экзаменационная ведомость';
if t1='к/пр' then form12.QRLabel1.caption:='Курсовой проект';
if t1='' then exit;
datamodule2.queryrep1.sql.clear;
```

Сброс текущих SQL-команд, загруженных в запрос Queryrep1

```
datamodule2.queryrep1.sql.add('select * from predmet');
```

Добавление SQL команды 'select * from predmet' в запрос Queryrep1

```
datamodule2.queryrep1.sql.add('where kpred='+
datamodule2.izuchenie.fieldbyname('kpred').asstring);
datamodule2.queryrep1.open;
```

Осуществление запроса

```
form12.QRLabel11.caption:=
datamodule2.queryrep1.fieldbyname('predm').value;
```

Выводит в отчет название предмета

```
datamodule2.queryrep1.close;
```

Закрытие запроса queryrep1

```
datamodule2.queryrep1.sql.clear;
datamodule2.queryrep1.sql.add('select * from prepod');
datamodule2.queryrep1.sql.add('where kprep='+
datamodule2.izuchenie.fieldbyname('kprep').asstring);
datamodule2.queryrep1.open;
form12.QRLabel13.caption:=
datamodule2.queryrep1.fieldbyname('fioprep').value;
```

Выводит в отчет ФИО преподавателя

```
datamodule2.queryrep1.close;
datamodule2.queryrep1.sql.clear;
datamodule2.queryrep1.sql.add('select ng,sh_spec from gruppa');
datamodule2.queryrep1.sql.add('where ng='+''''+
datamodule2.izuchenie.fieldbyname('ng').asstring+'''');
datamodule2.queryrep1.open;
form12.QRLabel5.caption:=
datamodule2.queryrep1.fieldbyname('sh_spec').value;
```

Выводит в отчет шифр специальности.

```
datamodule2.queryrep1.close;
datamodule2.queryrep1.sql.clear;
datamodule2.queryrep1.sql.add('select * from student');
datamodule2.queryrep1.sql.add('where student.ng='+''''+
datamodule2.izuchenie.fieldbyname('ng').asstring+'''');
datamodule2.queryrep1.sql.add('order by nomer_stud');
datamodule2.queryrep1.open;
```

Выводит в отчет список студентов выбранной группы.

```
form12.QRLabel7.caption:=
datamodule2.izuchenie.fieldbyname('semestr').asstring;
```

Выводит в отчет номер семестра

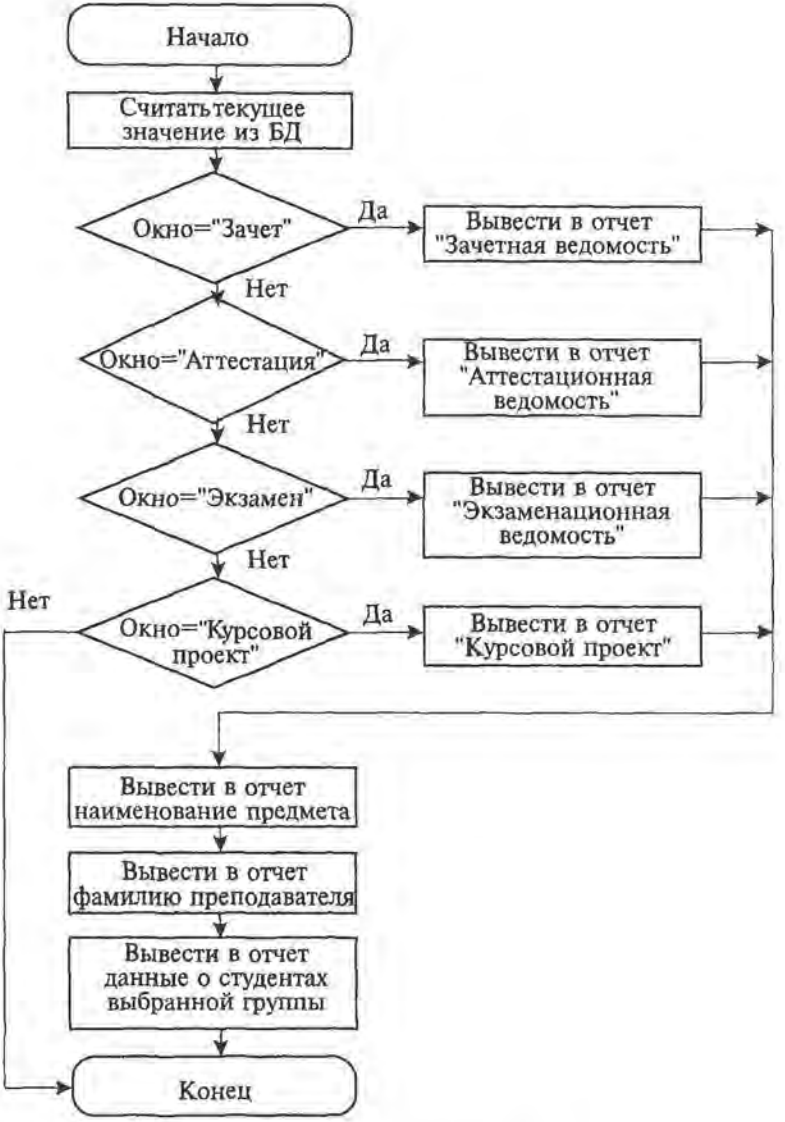


Рис. 14.20. Блок-схема процедуры, генерирующей «зачетную (экзаменационную) ведомость»

```
form12.QRLabel9.caption:=
datamodule2.izuchenie.fieldbyname('ng').asstring;
```

Выводит в отчет номер группы

```
form12.quickrep1.Preview;
```

Выводит отчет на предварительный просмотр

```
datamodule2.queryrep1.close;
end;
```

Данную процедуру можно представить в виде блок-схемы, изображенной на рис. 14.20.

На этом закончим описание распределенной БД «Учебный процесс».

Контрольные вопросы

1. Какие подходы к проектированию БД вы знаете? В чем их разница? Каковы последствия различия в подходах?
2. Какие режимы использования БД вы знаете?
3. Перечислите и дайте характеристику этапам создания и реализации БД.
4. В чем отличие многопользовательского режима от однопользовательского при проектировании БД? При эксплуатации БД?
5. Что такое «приложение»?
6. Перечислите этапы проектирования БД при традиционном подходе.
7. Каковы источники и способы получения данных для БД?
8. Почему для примеров выбраны СУБД Acees и InterBase?
9. Перечислите возможные способы заполнения данных.
10. Назовите составные части БД, постепенно формируемые при ее реализации.
11. Что такое «хранимая процедура», «триггер», «генератор»? Для чего они используются?
12. Как создаются таблицы в СУБД InterBase?
13. Как устанавливаются связи в СУБД InterBase?
14. Зачем нужен вид (View)?

Глава 15

Современный подход к проектированию и реализации баз данных

Современный подход (рис. 2.6) рассмотрим на примере 2.2 (см. гл. 2) процедуры приема на работу. При изложении подхода воспользуемся градацией гл. 15.

Реализация проводится на СУБД InterBase в среде Delphi.

Предполагается, что пользователь знаком с языком программирования Object Pascal (или Pascal), имеет некоторые навыки в работе с Delphi применительно к автоматизации программирования и знает начала интерфейсного и вложенного языков SQL.

Рассматривается режим работы клиент—сервер с использованием одноуровневой структуры. Первоначально изучается локальный вариант (сервер и клиент расположены на одном компьютере) с одним клиентом. Этот вариант используется чаще всего для отладки удаленного варианта (сервер и клиент — на разных компьютерах).

Далее рассматривается удаленный вариант и процедура перехода к нему от локального варианта. Сначала изучается случай с наличием одного клиента, а затем — с несколькими клиентами.

Отдельно рассматривается специфика реализации многоуровневой структуры режима клиент—сервер в среде Delphi.

15.1. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Анализ требований. Процедура приема на работу по анкетным данным достаточно трудоемка. В условиях ужесточающихся требований к работникам эту процедуру целесообразно выполнять в два этапа:

1) выработка с помощью компьютера (на основе принятых правил) решений-советов о приеме претендентов на работу;

2) принятие менеджером окончательного решения о принимаемых на работу.

Такой подход позволяет, с одной стороны, на первом этапе разгрузить менеджера от рутинных работ, оставив за ним творческие

процедуры, с другой стороны, на втором этапе — повысить оперативность введения в базу данных сведений о кадрах фирмы.

Сосредоточим внимание на первом этапе.

Для создания БД необходимо прежде всего исследовать алгоритм приложения. Практически речь идет о построении варианта экспертной системы реального времени (ЭСРВ).

В общем случае процедура выявления алгоритма приложения, как показано в [44], достаточно сложна и трудоемка. Упростим ее, полагая, что алгоритм приложения задан.

Проведем его описание.

Общая схема работы системы показана на рис. 15.1. Нетрудно видеть, что она является разновидностью структуры информационно-советующего режима (см. рис. 2.2).

Работа ЭСРВ проводится в дискретном времени, состоящем по умолчанию из трех интервалов времени, составляющих *сессию* функционирования системы. Каждый интервал времени называют также *циклом*.

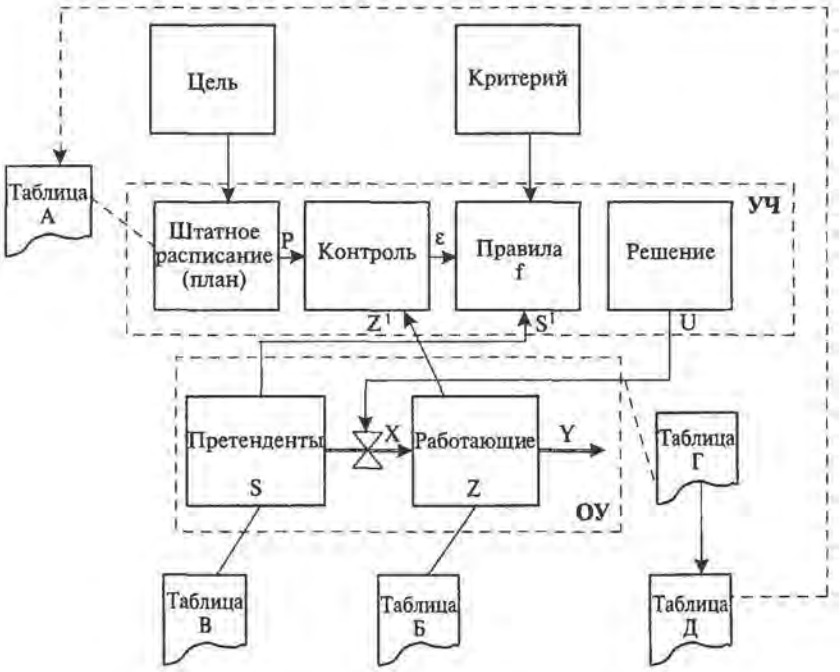


Рис. 15.1. Система управления приемом на работу

После окончания сессии система должна вернуться в исходное состояние.

Формально описание ЭСРВ представлено:

- описанием объекта управления;
- описанием действия среды, проявляющемся в увольнении людей и появлении претендентов в начале каждого цикла;
- математическим описанием дискретной (машина логического вывода) и непрерывной составляющей модели управляющей части;
- системой продукций (правил);
- объяснениями.

Для описания введем дополнительные обозначения: (t) — дискретные моменты времени $t \in [1, T]$, $[t] = [(t) - (t-1)] = \text{const}$ — интервал времени. Считаем, что на интервале $[t]$ скорости x, y, x', y', s, s' постоянны и меняются только на границе интервала времени. Пусть $[t]$ — один рабочий день.

Обозначим через $x[t]$, $y[t]$ — число принятых и уволенных за день. Очевидно, что x, y, s — векторы. Например, $x = \{x_1, x_2, x_3\}^{\mathrm{T}}$, т — признак транспонирования, $\{x_v, v \in [1, 3]\}$, $v = 1$ — научный сотрудник, $v = 2$ — инженер-конструктор, $v = 3$ — инженер по эксплуатации. Индексом $(')$ обозначена информация о соответствующих координатах: $x = x'$, $y = y'$, $s = s'$.

План P может задаваться двояко: ежедневный; с накоплением (например, с начала месяца).

Присвоим цифры словесно выраженным решениям r, суммарное количество которых $R\left(r \in \left(\overline{1, R}\right)\right)$ равно числу претендентов:

$i = 0$ — «отказать»,
$i = 1$ — «научный сотрудник»,
$i = 2$ — «инженер-конструктор»,
$i = 3$ — «инженер по эксплуатации».

Описание объекта управления имеет вид

$$z(t) = z(t-1) + [t](x[t] - y[t]). \qquad (15.1)$$

Уволенные (описание действия среды) определяются в режиме диалога в конце каждого цикла $[t]$.

Описание управляющей части определяется дискретной и непрерывной составляющими.

Дискретная составляющая представлена системой правил (табл. 15.1).

Формально система правил:

Таблица 15.1

Таблица правил

Номер правила	Ученая степень (A)	Открытия (B)	Знак балл	Средний балл (C)	Знак стаж	Стаж (D)	Должность
1	Нет	—	—	—	—	—	Отказать
2	Да	Да	—	—	—	—	Науч_сотр
3	Да	Нет	>=	3,5	—	—	Инж_конс
4	Да	Нет	<	3,5	>=	2	Инж_эксп
5	Да	Нет	<	3,5	<	2	Отказать

$$i_r[t] = \begin{cases} 0, & \text{A = нет;} \\ 0, & \text{A = да,} \quad \text{B = нет,} \quad \text{C} < 3{,}5, \ \text{D} < 2; \\ 1, & \text{A = да,} \quad \text{B = да;} \\ 2, & \text{A = да,} \quad \text{B = нет,} \quad \text{C} \geq 3{,}5; \\ 3, & \text{A = да,} \quad \text{B = нет,} \quad \text{C} < 3{,}5, \ \text{D} \geq 2, \end{cases} \tag{15.2}$$

где $r \in \overline{[1, R]}$, $i \in \overline{[1, 3]}$.

Непрерывная составляющая управляющей части системы состоит из уравнений:

$$w_{ri}[t] = w_{r-1,i}[t] + 1(i_r[t]), \text{ если } i_r[t] \neq 0, \ w_{0i}[0] = 0; \tag{15.3}$$

при решении u:

$$u_i[t] = w_{ri}[t] \text{ при } r = R \tag{15.4}$$

и

$$x[t] = u[t], \ u(t) \leq \varepsilon(t) \tag{15.5}$$

$$\varepsilon = p - z', \tag{15.6}$$

где p — план приема по штатному расписанию, $z' = z$ — информация о состоянии объекта управления (количестве работающих).

Объяснения. В процессе работы машины (логического) вывода на основе правил (15.2) компьютер вырабатывает решения-советы. Если у менеджера возникает сомнение в правильности решения-совета, он может запросить у компьютера объяснения, которые выдаются в словесной формулировке использованного правила (табл. 15.1).

Например, претендента Петрова рекомендовано принять научным сотрудником на основе правила 2, которое гласит:

ЕСЛИ *Ученая степень — Да* и *Открытия — Да*, ТО *принять научным сотрудником.*

Процесс работы пользователя показан на рис. 15.2.

На основе анализа алгоритма приложения возможно составить техническое задание

Техническое задание

Для выполнения проектных работ необходимы специалисты в соответствии со штатным расписанием. Если имеются вакантные места, отбор претендентов осуществляют по анкетным данным.

Необходимо спроектировать — в помощь руководителю — соответствующую экспертную систему реального времени. Предполагается, что отбор научных сотрудников осуществляет научное подразделение фирмы, а инженерного состава — производственное подразделение.

Объем БД не превышает 1 Гбайт.

Необходимо использовать широко распространенную СУБД. Интерфейс БД следует рассчитывать на конечного пользователя (КП) начального уровня, т. е. от КП не требуется знаний языков программирования высокого уровня.

БД должна иметь средства восстановления при сбоях и обладать достаточным быстродействием.

Концептуальная модель. На основе ТЗ можно составить набор таблиц БД. Таблица «Правила» приведена ранее. В таблицах «Штатное расписание» (табл. 15.2) и «Список работающих» (табл. 15.3) данные приведены только для первого цикла.

Таблица 15.2

Штатное расписание (начальное)

Время	Должность	План	Факт	Вакансии
1	Научный сотрудник	10	5	5
1	Инженер-конструктор	9	7	2
1	Инженер по эксплуатации	12	9	3

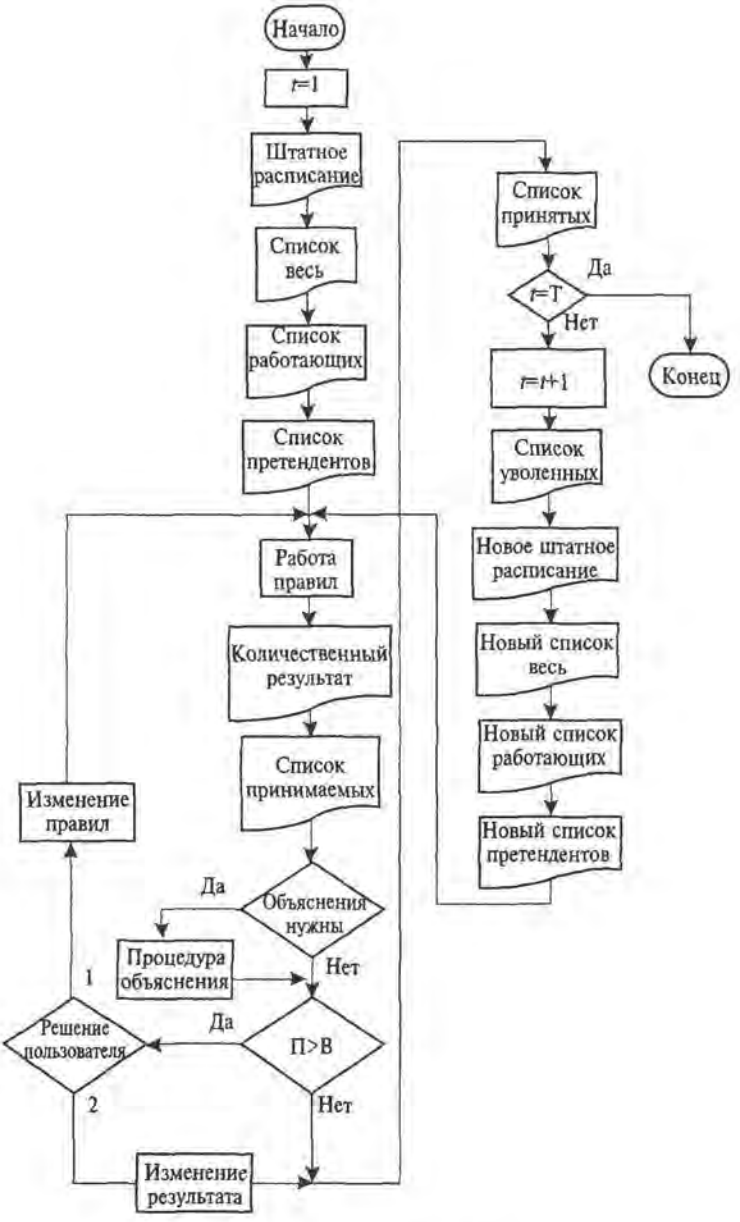


Рис. 15.2. Процедура работы с БД:
П — принимаемые; В — вакансии

Таблица 15.3

Список штатного расписания (начальное)

Время	Фамилия	Ученая степень	Открытия	Средний балл	Стаж	Статус	Должность
1	Водопьянов	Да	Да	4,8	7	Работ.	Науч. сотр.
1	Каримов	Да	Да	3,3	5	Работ.	Науч. сотр.
1	Крамской	Да	Да	4,2	8	Работ.	Науч. сотр.
1	Крикунов	Да	Да	3,5	9	Работ.	Науч. сотр.
1	Трубецков	Да	Да	4,1	15	Работ.	Науч. сотр.
1	Крымов	Да	Нет	4,0	4	Работ.	Инж.-конс.
1	Мамедов	Да	Нет	3,9	6	Работ.	Инж.-конс.
1	Орлов	Да	Нет	3,7	3	Работ.	Инж.-конс.
1	Синцов	Да	Нет	4,5	1	Работ.	Инж.-конс.
1	Петрович	Да	Нет	4,2	7	Работ.	Инж.-конс.
1	Тараканов	Да	Нет	4,1	3	Работ.	Инж.-конс.
1	Травкин	Да	Нет	5,0	9	Работ.	Инж.-конс.
1	Хохлов	Да	Нет	4,0	4	Работ.	Инж.-конс.
1	Черкас	Да	Нет	4,8	2	Работ.	Инж.-конс.
1	Касымов	Да	Нет	3,3	3	Работ.	Инж.-экспл.
1	Контуров	Да	Нет	3,0	4	Работ.	Инж.-экспл.

Продолжение табл. 15.3

Время	Фамилия	Ученая степень	Открытия	Средний балл	Стаж	Статус	Должность
1	Куприн	Да	Нет	3,3	5	Работ.	Инж.-экспл.
1	Ребров	Да	Нет	3,4	7	Работ.	Инж.-экспл.
1	Ремезов	Да	Нет	3,4	5	Работ.	Инж.-экспл.
1	Соколов	Да	Нет	3,3	3	Работ.	Инж.-экспл.
1	Тиунов	Да	Нет	3,0	6	Работ.	Инж.-экспл.
1	Троекуров	Да	Нет	3,0	4	Работ.	Инж.-экспл.
1	Щавель	Да	Нет	3,2	9	Работ.	Инж.-экспл.
1	Карпов	Да	Да	3,2	1	Претен.	
1	Крылов	Да	Да	3,1	2	Претен.	
1	Синцов	Да	Нет	4,5	1	Претен.	
1	Симонов	Да	Нет	3,9	2	Претен.	
1	Иванов	Да	Нет	3,2	3	Претен.	
1	Козлов	Да	Нет	3,4	4	Претен.	
1	Петров	Да	Да	4,0	1	Претен.	
1	Гуров	Да	Нет	4,0	3	Претен.	
1	Цветков	Да	Нет	3,0	1	Претен.	

Фрагментация и локализация. Из технического задания следует, что необходимо провести горизонтальную фрагментацию таблиц, что удобнее всего осуществить введением видов. Такой вариант, независимо от выбранной СУБД, является одновременно одним из способов защиты данных от несанкционированного доступа. Естественно, что БД располагается на сервере. Дублирование данных осуществим с помощью создания резервной копии.

Выбор СУБД. Остановим выбор, как и ранее, на реляционной модели данных, как широко применяемой.

Нетрудно видеть, что БД должна быть многопользовательской, что возможно достичь многопользовательским (файл-серверным) режимом или режимом клиент—сервер. Предпочтительным, в силу меньшего трафика, является режим клиент—сервер.

В качестве критериев выбора конкретной СУБД в соответствии с ТЗ могут быть использованы, как в п. 14.4, механизм блокировки, особенности хранения данных, производительность, характер инсталляции, обслуживание, конфигурирование и настройка, восстановление данных при сбоях, требование к ресурсам (см. табл. 14.1).

Перечисленным критериям в наибольшей степени удовлетворяет СУБД InterBase, используемая в среде Delphi.

Приведем первоначально основные предельные характеристики InterBase.

Объем БД — 10 Гбайт, количество полей — до 10 000, количество записей — не ограничено, число таблиц — до 65 000, длина записи — до 64 кбайт, длина поля — до 32 кбайт, вложенность SQL-запроса — до 16, размер хранимой процедуры или триггера — до 48 кбайт.

InterBase использует следующие типы данных: small (± 32 767, 2 байта); integer (± 2 млрд; 4 байта); float (± 3,4 E ± 38; 4 байта); double precision (± 1,7 E ± 308; 8 байт); char, varchar (символьный тип, до 255 символов); date. Вместо типа данных float лучше указывать decimal(a, b) или numeric(a, b), где a — количество знаков; b — число знаков после запятой.

Логическая модель. В силу специфики таблиц связи между ними, в виде схемы связей, не создаются. Необходимые связи устанавливаются в соответствующих запросах.

15.2. РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ

Отметим, что для проверки работы рассматриваемая база данных первоначально реализована и тестирована с использованием СУБД Access. Интерфейс пользователя БД выполнен с применением меню (рис. 15.3). Такой подход позволил не только отработать основные положения работы пользователя с БД, но и унифицировать алгоритм приложения с помощью системы используемых далее шаблонов.

Вместе с тем СУБД Access сталкивается, как отмечалось ранее, с серьезными проблемами создания сетевого варианта (распределенной БД), определяемого техническим заданием. В силу этого дальнейшее изложение касается СУБД InterBase в среде Delphi.

Рис. 15.3. Интерфейс пользователя, реализованный в СУБД Access

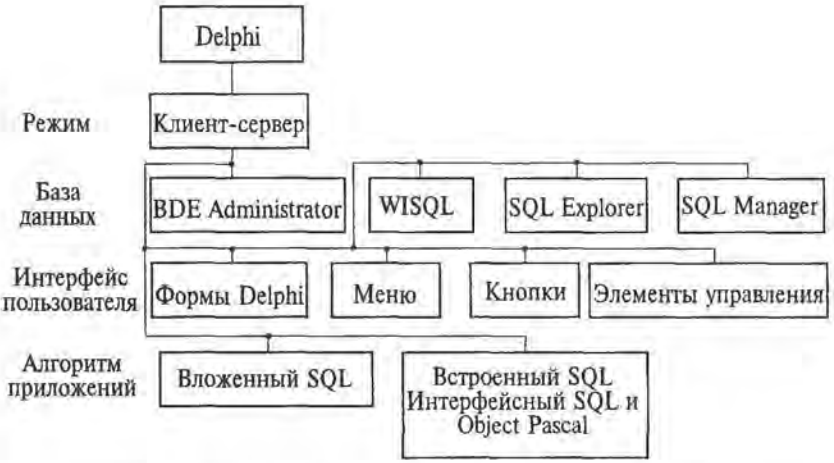


Рис. 15.4. Система реализации режима клиент—сервер

Система реализации БД в режиме клиент—сервер показана на рис. 15.4.

В режиме клиент—сервер выделяются два варианта; локальный (сервер и клиент расположены на одном компьютере) и удаленный (сервер и клиент — на разных компьютерах).

Рассмотрим последовательно названные варианты, при этом в локальном варианте процедуру фрагментации опустим.

Локальный вариант режима клиент—сервер

Режим клиент—сервер может быть построен с помощью:

1) прямых запросов к серверу (через компонент Query);

2) запросов через хранимые процедуры, «расположенные» на сервере.

Первый способ подробно описан в [19]. Для него характерна высокая вычислительная нагрузка клиентов, в силу чего такой способ удобнее назвать квазирежимом клиент—сервер.

Более грамотным — для реализации алгоритма приложения — является второй способ с использованием для хранимых процедур вложенного языка SQL. Рассмотрим его более подробно.

СУБД InterBase «работает» с вложенным языком SQL. В то же время запросы с клиентского приложения осуществляются с помощью интерфейсного SQL через компонент TQuery и языка программирования Object Pascal (возможно через формируемый SQL-запрос).

Нетрудно видеть, что среди типов данных нет счетчика (автоинкремент — в СУБД Paradox). Его заменяет специальная программа, получившая название *генератор*. Генератор обычно связывают с SQL-оператором или триггером, реже — с хранимой процедурой.

Не предусмотрено в InterBase и полуавтоматическое задание условий на значение, ссылочной целостности, как это имеет место в СУБД Paradox. Условия на значение и ограничения определяются ограничениями CONSTRAINTS, задаваемыми в неявном (без указания имени) или явном (с указанием имени) видах.

Для реализации остальных процедур используют дополнительные программы, называемые *триггерами*. Они «работают» с событиями Before* (до), After* (после) каких-либо изменений в таблицах. Генераторы, хранимые процедуры, триггеры пишутся на вложенном языке программирования SQL.

А. СОБСТВЕННО БД. В построении БД на сервере по-прежнему выделяется формирование алиаса и структуры БД, заполнение таблиц данными.

Алиас. Выберем в качестве имени алиаса priem_s. В СУБД InterBase все объекты помещаются в один файл. В связи с этим первоначально следует создать файл БД (с именем priems) с помощью утилиты WISQL. Ее запуск осуществляется через элемент Interbase 4.2/WISQL головного меню Windows.

В головном меню WISQL выберем File/Create Database и в появившемся окне Create Database установим локальный вариант InterBase (Local Engine), имя пользователя (путь, например, D:\student\Chert_cl\priems.gdb) и пароль (по умолчанию — masterkey) системного администратора для локального сервера InterBase.

Для надежной «руссификации» в окне Database Options следует указать DEFAULT CHARACTER SET WIN1251.

Нажатие кнопки *OK* приводит к запоминанию файла БД.

Задание имени БД возможно и программно — с помощью script-файлов, описанных позднее.

Теперь можно — для формирования алиаса — обратиться к утилите BDE Administrator. В ее головном меню через Object/New выберем в окне New Database Alias в качестве имени драйвера INTRBASE (INTERBASE1). Заменим его на priem_s. В правом окне в строке SERVERNAME укажем путь к созданному файлу. В строке USERNAME зададим имя SYSDBA. Щелкнем правой кнопкой мыши на имени алиаса в левом окне и выберем элемент меню Apply и зафиксируем созданный алиас. Нажмем «+» слева от имени алиаса и в открывшемся окне введем пароль masterkey.

Отметим, что имя пользователя и пароль являются средствами защиты БД и при работе InterBase будут часто запрашиваться. На время отладки БД такой запрос можно заблокировать.

Создание алиаса закончено.

Его можно использовать непосредственно в компоненте Query одноименной цепочки доступа к БД. Однако предпочтительнее — для улучшения процесса управления БД — перед объектом Query в головной форме Delphi вставить компонент Database (обычно один для базы данных). В нем в качестве свойства AliasName установим priem_s, а в качестве свойства DataBaseName — priem_ss. Теперь алиасом для компонент Query станет priem_ss.

Создание структуры таблиц. Структура таблиц создается через программу, написанную на вложенном языке SQL. Ее можно вводить в компьютер двумя способами.

1. Первоначально написать всю программу (script-файл) в любом из текстовых редакторов. При ее сохранении файлу следует задать расширение sql. Затем программу можно ввести с помощью утилиты WISQL (Run on ISQL Script).

Такой способ удобен при наличии повторяющейся последовательности SQL-операторов и при использовании отлаженных файлов. При их формировании требуется большая внимательность, поскольку при наличии ошибок в программе первый способ фактически сводится ко второму.

2. Здесь может использоваться утилита WISQL или утилита SQL Exlorer.

При вызове WISQL полезно закрепить «русификацию». Для этого в режиме Session/Advanced Setting в окне Character Set On надо задать WIN1251.

Для создания структуры таблиц и их заполнения используется язык SQL.

Установим сетевое соединение БД с WISQL через элемент File/Connect to Database его головного меню. В открывшемся окне зададим путь к БД, имя пользователя и пароль.

В появившейся заставке в верхнем окне SQL Statement следует набрать SQL-оператор. Так, для создания таблицы Pravila наберем

```
CREATE TABLE Pravila(
Nomprav integer Not Null,
FIO varchar(15),
Stepen varchar(5),
Otkrytie varchar(5),
```

```
Znak_ball varchar(5),
Sr_ball numeric(15,2),
Znak_Stag varchar(5),
Stag integer,
Dolgnost varchar(10),
Objasnenia varchar(120)
);
```

По завершении набора следует нажать кнопку *Run*.

Набор «переходит» в окно ISQL Output и при отсутствии ошибок выполняется. При наличии ошибок выдается сообщение, в котором указывается строка и знак в наборе, где допущена ошибка. После ее исправления следует снова нажать кнопку *Run*.

Описанная процедура утомительна, и потому в WISQL кнопками *Previous* и *Next* можно возвратить необходимый оператор из нижнего окна в верхнее. После его корректировки он вводится нажатием кнопки *Run*.

Подтверждение окончания набора программы осуществляется выбором в головном меню File/Commit Work, а отказ от выполнения — File/Rollback Work.

После завершения работ с WISQL отсоединение от БД проводится выбором File/Disconnect from Database. Выбор кнопки *Exit* означает выход из WISQL.

Следует отметить, что вложенный SQL имеет слабые средства отладки программ и неудобный интерфейс. Однако высокое быстродействие программ при работе в сетевом режиме заставляет мириться с этим недостатком.

Чтобы контролировать работу WISQL, возможно использовать меню.

При нажатии элемента меню View/Metadata Information выдается заставка View Information. В ней можно выбрать тип и имя объекта, в результате чего в окне ISQL Output будет показана программа объекта на вложенном языке SQL.

Если имя объекта не задано, выдается список объектов данного типа.

Выбор элемента меню Extract /SQL Metadata for Table и имени таблицы приведет к выдаче данных о ней, которые могут быть сохранены в txt-файле.

Нажатие Extract /SQL Metadata for Database вызовет выдачу метаданных о БД в виде программы на вложенном языке SQL. Ее можно скопировать в буфер и затем сохранить, например, в редакторе Word.

Пример такой SQL-программы приведен в Приложении 3.

Заполнение таблиц. Заполнение можно осуществить в WISQL или в SQL Explorer.

В первом случае производится построчная вставка данных с помощью оператора INSERT. Корректировка данных может осуществляться операторами UPDATE и DELETE, контроль полученных результатов — оператором SELECT. Однако такой способ утомителен.

Более удобно использование утилиты SQL Explorer, которая позволяет к тому же создавать структуру таблиц.

После открытия SQL Explorer на экране появляется двухоконная заставка. В левом окне выбирается закладка Databases и необходимый алиас. Нажатие «+» слева от него раскрывает перечень типов объектов БД. Нажатие «+» у выбранного типа объектов раскрывает перечень имен этого типа объектов. Выбор имени вызывает появление в правом окне трех (для таблиц — четырех) закладок.

Закладка Definition определяет метаданные конкретного объекта, закладка Text позволяет получить SQL-оператор создания конкретного объекта. Выбор закладки Data (для таблиц) позволяет провести просмотр, заполнение и корректировку данных в выбранной таблице.

Для введения новых данных следует нажать «+» в навигаторе и последовательно вводить записи. При переходе на последующую запись предыдущая запоминается автоматически. Для запоминания последней записи необходимо нажать кнопку *Post* или *Refresh* навигатора.

Создание программного кода любого объекта осуществляется при выборе закладки Enter SQL. В верхней части правого окна набирается SQL-оператор и нажимается кнопка с изображением молнии. Если оператор набран корректно, в нижней части правого окна появляется результат выполнения оператора (в противном случае выдается сообщение об ошибке). После успешного выполнения оператора следует обновить в утилите информацию о БД с помощью элемента меню View/Refresh.

Контроль за работой БД и сервера в удаленном варианте осуществляется утилитой Interbase Server Manager.

Б. ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ. В режиме клиент—сервер серверная часть интерфейса пользователя развита, как правило, слабо. В нем выводятся параметры, общие для всех клиентов. Таких параметров обычно немного.

Основной акцент в разработке интерфейса пользователя переносится на интерфейс клиента.

При формировании интерфейса учтем опыт, накопленный при выполнении работы [19], где интерфейс пользователя построен на

основе одной формы Delphi. В связи с этим усложнился программный код за счет организации доступа к элементам управления, введения дополнительных кнопок при переходе от групп элементов меню к другим элементам меню, постоянного изменения SQL-оператора в компонентах Query.

При использовании нескольких форм компоненты Query можно закрепить за конкретными выводимыми на экран вариантами таблиц, переносом SQL-операторов в свойства SQL компонент Query. При этом потребуется дополнительный программный код для взаимодействия форм, но он много проще «устраняемого» программного кода.

В общем случае можно использовать девять форм Delphi для клиента и одну — для сервера.

При переходе к хранимым процедурам алгоритм приложения клиентской части резко упрощается. Для таблиц «Кадры», «Работающие», «Претенденты», «Принимаемые», «Принятые» SQL-операторы почти совпадают.

В связи с этим лучше использовать четыре формы Delphi для клиента и одну — для сервера. Тогда возможно такое «распределение» таблиц.

Сервер

Form5, на которой расположены компоненты Edit1 и Edit2 для фиксации текущего цикла и общего количества циклов.

Клиент

Form1 (основная) — меню и таблица «Штатное расписание».
Form2 — таблица «Кадры» и ее составные части.
Form3 — таблица «Объяснения».
Form4 — таблица «Правила».

Формы Delphi Form2 — Form4 — модальные. При их закрытии осуществим переключение доступных элементов головного меню.

В головном меню Form1 выделим элементы *Главная*, *Показать*, *Обработать*.

Для элементов головного меню предварительно введем следующее деление.

Главная

Открыть
Выйти

Показать

Штатное расписание

Кадры
Претенденты
Работающие

Принимаемые

Принятые

Уволенные

Обработать

Начать

Запуск правил

Количественный результат
Объяснения
Изменение правил
Изменение результата

Принять всех

Продолжить

Замечание. При использовании нескольких форм следует установить необходимые связи между ними. Для этого в модуле unit после заголовка implementation пишется строка

```
uses unit1, ..., uniti, ...unitm;
```

Без установления связей формы взаимодействовать не будут.

В. АЛГОРИТМ ПРИЛОЖЕНИЯ. Для формирования алгоритма используем шаблоны [19]. *Под шаблоном* понимается законченная в целевом отношении и повторяющаяся программная часть процедуры (procedure) или функции (function). Шаблон может быть построен из элементарных и комбинированных составляющих.

ВП1. ***Элементарные составляющие.*** Простейшими (элементарными) составляющими являются операторы SQL: выборки (SELECT), обновления (UPDATE, DELETE, INSERT), уничтожения (DROP) и построения (CREATE) таблиц (TABLE).

Следует отметить возможность двух способов их реализации.

1. Использование свойства SQL — компонента TQuery:

```
select * from kadry
```

Этот способ применяют при многоформном (многооконном) интерфейсе, когда объект Query формы «закреплен» за конкретной таблицей и вид запроса не меняется в течение всей сессии.

2. Составление формируемого запроса, размещаемого в программном коде. Тогда предыдущий пример получит вид

```
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('select * from kadry');
Query1.Open;
```

Этот способ применяется при однооконном варианте интерфейса или в случае частого изменения вида запроса в объекте Query1.

С точки зрения написания SQL-запроса разница невелика, поэтому дальнейшую процедуру формирования шаблонов иллюстрируем с использованием SQL-свойств.

ВП1.1. ***Оператор select*** используется в многочисленных вариантах, определяемых следующими основными признаками:

а) использованием параметров (два класса — обычный и с параметрами);

б) применением функций агрегирования (обычный, агрегированный);

в) возможностью редактирования табличных данных после вызова таблицы на экран (обычный, RequestLive);

г) использованием одной или нескольких исходных таблиц.

Для проектируемой БЗ представляют интерес следующие варианты.

1. Простая выборка (будем называть вариант **select1**):

```
select * from kadry
```

2. Выборка с параметрами (select2), в которой выделим случаи:

- с одной таблицей (select21):

```
select dolgnos, count(dolgnos) as fakty from kadry
where (status='работ')
And(vremja between 1 And :vremja)
group by dolgnos
ParamByName('Vremja').Value:=StrToInt(Edit1.Text);
```

- с двумя таблицами (select22):

```
select K.vremja, K.Familia,
```

```
K.Uch_stepen, K.Otkrytie, K.Sr_ball,
K.Stag, K.dolgnos,
P.Objasnenia from kadry K, pravila P
where vremja=:vremja And
P.nomprav=K.nomprav
Vremja').Value:=StrToInt(Edit1.Text)
```

3. Выборка с вычислениями и функциями агрегирования (select3):

```
select dolgnos, count(dolgnos) as fakty from kadry
where status='работ'
group by dolgnos
```

4. Выборка с параметрами и возможностями изменения вызванных данных в диалоге:

```
RequestLive:=True;
select * from kadry
where (status='приним')
and (vremja=:vremja)
ParamByName('Vremja').Value:=StrToInt(Edit1.Text);
```

ВП1.2. ***Оператор update*** с тремя разновидностями:

1. С параметрами update1, включающий два подкласса:

- имя изменяемого поля не входит в условия (update11):

```
Update kadry
set nomprav=:nomprav,
dolgnos=:dolgnos
where (uch_stepen=:uch_stepen)
And (otkrytie=:otkrytie)
And (Sr_ball<:Sr_ball)
And (Stag>=:Stag)
And  (status='претен')
And (vremja=:vremja)
ParamByName('nomprav').Value:=Query2.FieldByName('nomprav').Value;
```

- имя изменяемого поля входит в условия (update12), при этом для переменной используется приставка old_

```
Update kadry
set status=:status
```

```
where (status=:old_status)
And (vremja=:vremja)
ParamByName('status').Value:='принят';
```

2. С параметрами, вычислениями и использованием цикла (update2):

```
Update stat_rasp
set fakty=:fakty,
vacanc=plany -fakty
where (dolgnos=:dolgnos)
And (vremja=:vremja)
ParamByName('Vremja').Value:=StrToInt(Edit1.Text);
```

3. С параметрами, циклом и ссылкой Datasource на параметры (update3):

```
update stat_rasp
set plany=:plany, fakty=:fakty,
vacanc=:vacanc, prinimaem=:prinimaem,
nedobor=:nedobor
where vremja=:vremja and dolgnos=:dolgnos
Datasource:=DataSource2;
```

ВП1.3. ***Оператор insert*****:**
1. Обычный (insert1):

```
insert into kadry
    values('Петров', 'да', ...);
```

ВП1.4. ***Оператор delete*****:**
1. Обычный (delete1):

```
delete from pravila
```

2. С параметрами:

```
delete from kadry
where vremja=:vremja and dolgnos='отказать'
ParamByName('Vremja').Value:=StrToInt(Edit1.Text);
```

ВП1.5. ***Оператор drop*** уничтожения таблицы (drop1)

```
drop table pravila
```

ВП1.6. ***Оператор create*** создания таблицы (create1), описанный ранее.

ВП2. ***Комплексные составляющие, использующие не менее двух элементарных составляющих.***

ВП2.1. ***Составляющая insertI*** (insert1 + select1):

```
insert into kadry
select * from kadry_ish
```

ВП2.2. ***Составляющая updateI*** (update1 + select3):

```
    update stat_rasp
        set plany=:plany, fakty=:fakty,
        vacanc=:vacanc, prinimaem=:prinimaem,
            where (vremja=:vremja) and
                dolgnos IN
(select dolgnos, count(dolgnos) as fakty
        from kadry
where (status='работ') and
        (vremja between 1 and :vremja)
    group by dolgnos)
ParamByName('Vremja').Value:=StrToInt(Edit1.Text);
```

ВП3. На основе этих составляющих формируются шаблоны, показанные в табл. 1.1 и «привязанные» к действиям пользователя — нажатию элементов меню (см. рис. 15.2.).

При построении алгоритма приложения шаблоны связываются с соответствующими элементами меню и апробируются. Однотипные шаблоны могут просто копироваться.

Отметим одно интересное обстоятельство. Формирование программных шаблонов (табл. 15.4) фактически осуществляется во всех СУБД, но в различных вариантах.

В СУБД Access составляющими являются запросы, в которых хранятся и промежуточные результаты. В СУБД Paradox (в рамках Delphi) составляющие связаны с компонентами Query, которые хранят промежуточные данные. В СУБД InterBase контейнерами для шаблонов служат хранимые процедуры, в которых последовательно размещаются необходимые составляющие. Промежуточные результаты хранятся в предложении INTO программы хранимой процедуры в виде параметров.

Схема взаимодействия сервера и клиента в части алгоритма приложения показана на рис.15.5.

Таблица 15.4

Программные шаблоны алгоритма приложения

Действия пользователя	Шаблон
Начать	А-копирование Delete1 Insert1 или Б-копирование Select1 Update3 или В-копирование Update1 или Г-копирование Drop1 Create1 Insert1
Штатное расписание	Select3 Update2 Select21
Кадры	Select21
Работающие	Select21
Претенденты	Select21
Запуск правил	Select1 Update11 Update11
Принимаемые	Select21
Количественный результат	Select3 Update2 Select21
Объяснение	Select22
Принять всех	Delete2 Update12
Принятые	Select21
Продолжить	Update12 Б-копирование Select1 Select1

Продолжение табл. 15.4

Действия пользователя	Шаблон
Уволенные	Select4 Select1
Изменение результата	Select4
Изменение правил	Select21 Select4 Update12

В рассматриваемой программной реализации генераторы и триггеры не используются.

Фрагмент-пример программы алгоритма приложения приведен в приложении 3. В ней выделена серверная часть (на вложенном языке SQL) и клиентская часть (интерфейсный язык SQL и Object Pascal с 15 формами).

Клиентская часть достаточно проста. Она в значительной мере представляет собой программу интерфейса пользователя, поскольку почти все select-запросы «спрятаны» в SQL-свойства компонент Query.

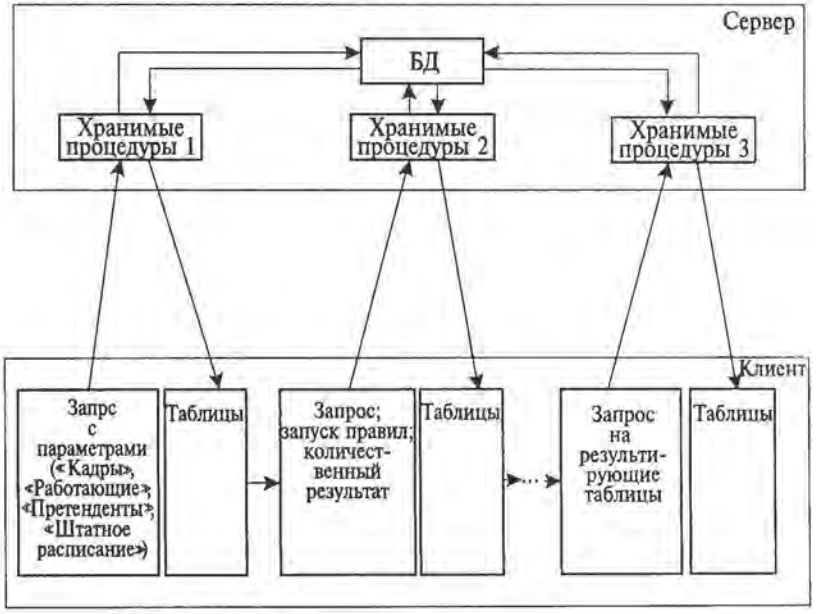


Рис. 15.5. Взаимодействие сервера и клиента

Программный код для сервера (код хранимых процедур) соответствует программным шаблонам, рассмотренным в табл. 15.4.

Программный код клиентской части также состоит преимущественно из кода интерфейса пользователя и кода алгоритма приложения. Последний представлен операторами типа select21, в которых имена таблиц заменены на имена хранимых процедур.

Подчеркнем еще раз, что и клиент, и сервер для локального варианта реализуются на одном, основном компьютере.

Удаленный вариант режима клиент—сервер

Удаленный вариант режима клиент—сервер может быть реализован в виде одно- и многоуровневой структуры.

Первоначально рассмотрим реализацию **одноуровневой** структуры (рис. 15.6) путем перехода от локального варианта к удаленному.

Каждый пользователь-клиент должен иметь свой интерфейс. В этом случае необходимы также процедуры фрагментации и локализации данных. Здесь же возникают вопросы взаимодействия клиентов и защиты их данных.

Для целей отладки БД вводят две фазы реализации одноуровневой структуры.

А. Имеет место только один клиент (один интерфейс).

Б. В приложении имеется несколько клиентов.

♦ **А.** На этой фазе серверную часть программы оставляют на основном компьютере, а клиентскую визуально-программную часть переносят на другой компьютер.

При реализации БД выполняется такая последовательность операций.

Рис. 15.6. Одноуровневая структура удаленного варианта режима клиент—сервер

1. Инсталлируется на сервере Delphi и удаленный вариант InterBase.

2. На компьютере-клиенте устанавливаются:

а) протокол доступа к InterBase (чаще всего — TCP/IP) в файле SERVICES (папка WINDOWS)

```
gds_db 3050/tcp;
```

б) IP-адрес сервера в файле HOSTS (папка WINDOWS), состоящий из цифрового кода и имени сервера, например

```
192.196.10.12 Server
```

3. На клиенте ставится Delphi и (только для процедуры создания БД) частично устанавливается InterBase (WISQL, InterBase Server Manager (IBSM)).

4. Проводится переустановка алиаса:

а) с помощью серверной утилиты BDE Administrator (далее — BDE) уничтожается локальный алиас;

б) в клиентском BDE проверяется наличие соответствующего драйвера из набора SQL Links;

в) с клиентского BDE устанавливается новый алиас. В его пути к БД имя диска заменяется на имя сервера: 195.201.72.76:d\... .

5. Через клиентскую утилиту WISQL (или утилиту IBSM) проводится соединение клиента с БД на сервере.

6. На клиенте устанавливается клиентская визуально-программная Delphi-часть приложения. В свойстве AliasName компонента Database выбирается вновь построенный алиас. В свойстве DataBaseName устанавливается другое имя, служащее алиасом для компонент Query (равно как и для Table, DataModule).

Теперь клиент может запустить программу в Delphi и начать работу.

Отметим особенности «введения» новых пользователей. В качестве параметров входа в БД могут быть использованы либо параметры системного администратора (user — SYSDBA, password — masterkey), либо другие параметры.

В последнем случае первоначальный доступ осуществляется через параметры системного администратора, а затем вводятся другие параметры. Такой прием используется для дополнительной защиты данных.

Возможна регистрация пользователей и с другими параметрами (например, DIMA и whr соответственно). Такие пользователи смо-

гут только подсоединиться к БД, однако не получат права доступа ни к одному объекту БД. В этом случае права доступа задаются с помощью операторов привилегий.

Так, для таблицы stat_rasp («Штатное расписание») и хранимой процедуры proc_statrasp, обращающейся к этой таблице, задание привилегий возможно в такой форме

```
GRANT ALL ON stat_rasp TO DIMA, (A)

GRANT EXECUTE ON PROCEDURE proc_statrasp TO DIMA. (B)
```

Если до получения доступа (B) к хранимой процедуре доступ (A) не был задан, то в дополнение к оператору (B) следует задать

```
GRANT ALL ON stat_rasp TO PROCEDURE proc_statrasp. (C)
```

Если из таблицы stat_rasp сформировать соответствующие виды для каждого пользователя и обеспечить (через операторы привилегий) доступ к этим видам, то таблица stat_rasp будет фрагментирована. При этом каждый пользователь получит доступ только к «своим» данным.

После успешной апробации для процедуры использования БЗ на компьютере-клиенте удаляются утилиты WISQL, IBSM.

♦ **Б.** При переходе ко второй фазе необходимо учесть фрагментацию, выполненную ранее.

В примере имеются два подразделения: научное и производственное. Научное подразделение заинтересовано в принятии научных сотрудников, производственное — в принятии инженеров. Таким образом, имеем два клиента-пользователя.

Очевидна фрагментация (деление по узлам, пользователям) таблиц «Штатное расписание», «Кадры», «Правила». При локализации (деление по задачам) не будем использовать копии фрагментов. Тогда результаты фрагментации и локализации совпадают.

Доступ к данным различным клиентам обеспечим с помощью формирования видов (View) для таблиц «Штатное расписание», «Кадры», «Правила». Из-за горизонтального разделения данных использование привилегий доступа GRANT проблематично: использование привилегий для определенных столбцов лишит клиента возможности изменять виды, что необходимо в соответствии с алгоритмом работы пользователя.

Чтобы в дальнейшем можно было вносить изменения в виды (просмотры), необходимо соблюдать следующие условия:

а) просмотр должен содержать *все* поля исходной таблицы;

б) исходная таблица для вида должна быть только одна;

в) оператор SELECT просмотра не должен использовать функций агрегации данных, предложения HAVING, соединения таблиц, хранимых процедур, функций, определенных пользователем.

Таким видом для таблицы «Кадры» (научное подразделение) может быть

```
CREATE VIEW Kadry_nauch
AS
SELECT * FROM Kadry
WHERE (dolgnos='науч_сотр' or dolgnos='отказать' or dolgnos
IsNull);
```

Теперь хранимая процедура для клиента (научное подразделение) обращается к виду Kadry_nauch.

При наличии нескольких клиентов необходимо рассмотреть два вопроса:

1) формирование новых пользователей;

2) выбор системы блокировки данных.

1. Обычно первоначально обращаются к пользователю-администратору БД с именем SYSDBA и паролем masterkey, которые впоследствии могут быть изменены.

Это делается с помощью утилиты InterBase Server Manager. В головном меню выбирают элемент Tasks/User Security. В появляющемся окне фиксируется система зарегистрированных на сервере пользователей. С помощью кнопок Add User, Modify User, Delete User этого окна возможно добавить, изменить, уничтожить имя и пароль пользователя.

Заметим, что утилита IBSM осуществляет сбор статистики, принудительную сборку мусора, создание резервной копии, восстановление БД из резервной копии, принудительную запись на диск.

2. При работе нескольких клиентов возникает вопрос одновременного изменения данных. Для его решения существует несколько уровней изоляции транзакций (TransIsolation): Dirty Read, Read Commited, Repeatable Read.

Дело в том, что в Delphi существует две версии данных: до транзакции изменения и после транзакции изменения.

При уровне Dirty Read конкурирующие транзакции изменения видят изменения, внесенные предыдущей транзакцией (1-я фаза), но не подтвержденные (2-я фаза). Если предыдущая транзакция

использует откат, то другие транзакции будут видеть недостоверные данные. Такой уровень изоляции используется редко.

При уровне изоляции Read Commited конкурирующие транзакции работают только с подтвержденными изменениями.

Уровень изоляции Repeatable Read дает возможность видеть данные в том состоянии, в котором они находились при старте транзакции.

Уровень изоляции устанавливается свойством TransIsolation компоненты Database. Чаще всего в InterBase используют уровень Read Commited.

Если две транзакции конкурируют на одной записи, «срабатывает» более ранняя в подтверждении транзакция.

Возможны три варианта разрешения такого конфликта, определяемые в компоненте Database UpdateMode:

WhereAll (по умолчанию) — поиск записи, которая должна быть изменена, по всем полям;

WhereKeyOnly — поиск по ключевым полям;

WhereChanged — поиск по ключевым полям и тем неключевым полям, которые были изменены при корректировке данных.

Отдельно следует сказать о кэшированных (отложенных) изменениях.

Изменения на клиенте могут накапливаться и лишь затем передаваться на сервер, что уменьшает вероятность конфликта транзакций.

Для кэширования следует в компоненте TQuery установить свойство

```
CashedUpdate=True
```

до запуска программы или во время ее выполнения. Сделанные кэшированные изменения могут быть отменены в компоненте Database методом CancelUpdates. Подтверждение изменений осуществляется методом ApplyUpdates. Для фиксации изменений в БД используется метод CommitUpdates.

Свойство UpdateRecordTypes позволяет указать тип «видимой» записи (измененная, добавленная, удаленная, неизмененная).

Ошибки, возникающие при подтверждении кэширования, можно учесть событием UpdateError формы Delphi.

В случае, когда результаты работы оператора SELECT доступны только для чтения, вместе с компонентом TQuery используют компонент TUpdateSQL.

Перейдем к **многоуровневой** структуре удаленного варианта режима клиент—сервер (рис. 15.7).

Рис. 15.7. Удаленный вариант режима клиент—сервер: многоуровневая структура

В одноуровневой структуре процедуры распределения данных, координация работы клиентов и защита данных выполняется на сервере. Клиент осуществляет запросы на сервер и получает ответы, для чего требуется достаточно сложное и объемное программное обеспечение.

Чтобы его упростить и уменьшить объем приложения, используют так называемую *многоуровневую* (в ряде литературных источников — трехуровневую) структуру удаленного варианта режима клиент—сервер (рис. 15.7).

Между сервером (теперь это сервер базы данных) и клиентами размещается сервер приложений. Он берет на себя функции обеспечения взаимодействия клиентов, защиту данных, что позволяет существенно упростить программное обеспечение сервера БД, который только хранит данные. Для всех клиентов устанавливается только одна утилита BDE на сервере приложений, что резко упрощает структуру и уменьшает объем памяти для клиентов. Сервер БД и сервер приложений смогут устанавливаться и на одном компьютере.

Для программного обеспечения многоуровневой структуры следует «научить» программы, как предоставлять другим программам свои функции (службы) и получать доступ к другим программам независимо от их расположения.

Для общего случая разработана базовая технология многокомпонентной модели объектов (Component Object Model — COM).

Для баз данных на основе COM предложена специальная технология — MultiTiered Distributed Application Services (MIDAS) или служба многоуровневого распределения приложения.

Ее достоинством является не только функциональная специализация функционально-аппаратных средств, но и возможность использования на клиентской стороне гетерогенных программно-аппаратных платформ.

MIDAS-технология обеспечивается страницей MIDAS палитры компонент Delphi и реализуется следующими прикладными технологиями.

Таблица 15.5

Сравнительные характеристики прикладных технологий

Технология	Достоинства	Недостатки
DCOM	Органичная связь с Windows Простота реализации	Работа только под Windows NT Нет координации серверов приложений Нет координации данных
OLEnterprise	Возможность работы с Windows 95 Координация серверов приложений *Переключение клиентов на* другие серверы	Нет в Delphi 5
MTS	Возможность работы с перечисленными технологиями Разграничение прав доступа на основе ролей пользователей и данных	Приобретается отдельно
CORBA	Гетерогенные операционные системы Несколько серверов приложений с переключением клиентов Дополнительная защита данных	Сложная структура

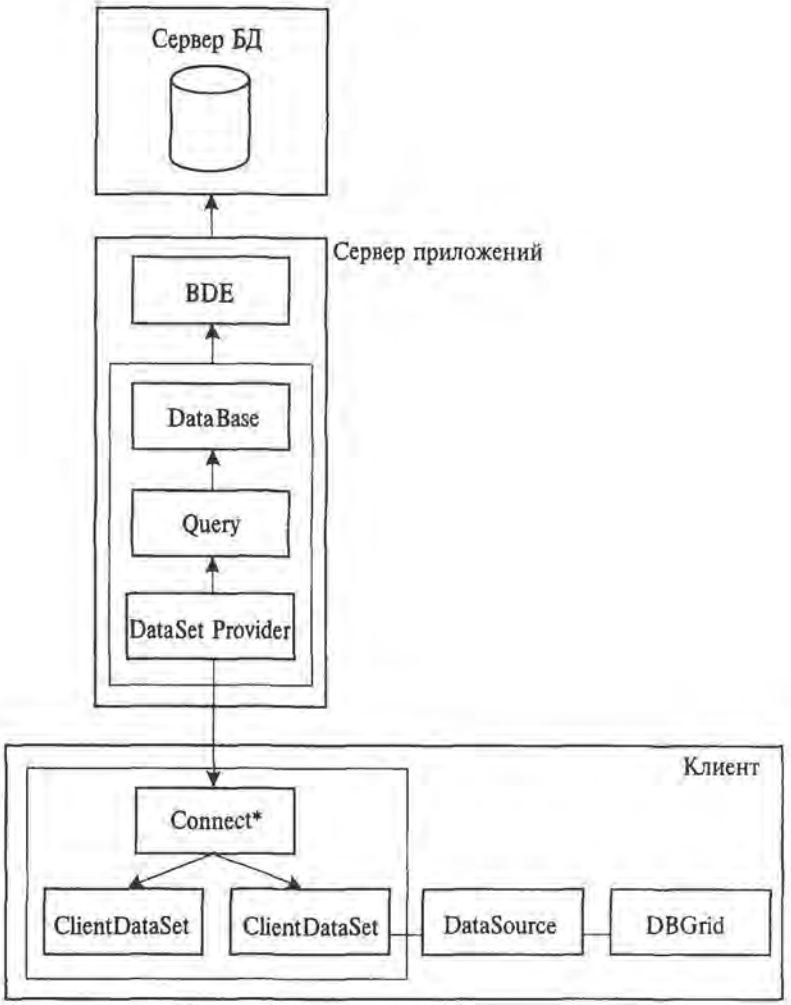


Рис. 15.8. Детализированная многоуровневая структура режима клиент—сервер:

* — содержание зависит от применяемой прикладной технологии

Distributed COM (DCOM) — расширение технологии COM, которая имеется в WindowsNT;

Сокеты TCP/IP, которые могут работать и без WindowsNT;

OLEnterprise, разработанная фирмой Borland;

Microsoft Transaction Server (MTS) — управления транзакциями, основанная на COM с повышенной помехозащищенностью;

Common Object Request Broker Architecture (CORBA) — архитектура брокера общих запросов для взаимодействия с данными разных операционных систем.

Заметим, что первые четыре прикладные технологии основываются на базовой технологии COM, которая первоначально была разработана для связи файлов на одном компьютере для одного и того же или для разных процессов. Позднее технология была доработана для связей файлов на разных компьютерах.

В последнем случае была разработана базовая технология CORBA, предназначенная изначально для удаленной связи файлов, реализованных в разных операционных системах. В силу более жестких требований технология CORBA имеет более сложную структуру, чем технология COM, но обладает большей защищенностью данных.

Трудно однозначно рекомендовать одну из перечисленных прикладных технологий, поскольку все зависит от требований, предъявляемых к удаленному варианту. Поэтому в табл. 15.5 приведены сравнительные характеристики прикладных технологий.

Детализированная схема многоуровневой структуры имеет вид, приведенный на рис. 15.8.

Контрольные вопросы

1. Как описывается объект управления?
2. Как описывается система правил? управляющая часть?
3. Можно ли использовать в интерфейсе пользователя меню? кнопочные формы?
4. Назовите последовательность этапов перехода от локального к удаленному варианту режима клиент—сервер (одноуровневая структура).
5. Назовите цель построения многоуровневой структуры.

ЗАКЛЮЧЕНИЕ

Базы данных остаются динамично развивающимся теоретико-прикладным направлением, при этом особенно прогрессирует прикладная сторона.

Ее развитие идет по следующим основным направлениям:

- разработка объектно-ориентированных СУБД и расширение масштабов применения объектно-ориентированных БД;
- создание сетей баз данных, в том числе корпоративных, intranet — с использованием принципов построения Internet;
- совершенствование режима клиент—сервер и сетевой связи серверов;
- построение хранилищ данных;
- увеличение памяти для БД за счет третичной составляющей (магнитные ленты, оптические диски) памяти.
- переход от традиционных баз данных, хранящих числа и символы, к объектно-реляционным базам данных, содержащим данные со сложным поведением.

Быстрое развитие технологий хранения информации, коммуникаций и обработки позволяет переместить всю информацию в киберпространство. Программное обеспечение для определения, поиска и визуализации оперативно доступной информации ключ к созданию и доступу к такой информации. Основные задачи, которые при этом необходимо решить, следующие:

- определение моделей данных для новых типов (например, пространственных, темпоральных, графических) и их интеграция с традиционными системами баз данных;
- масштабирование баз данных по размеру (до петабайт), пространственному размещению (распределенные) и многообразию (неоднородные);
- автоматическое обнаружение тенденций данных, структур и аномалий (добывание данных, анализ данных);
- интеграция (комбинирование) данных из нескольких источников;
- создание сценариев и управление потоком работ (процессом) и данными в организациях;
- автоматизация проектирования и администрирования базами данных.

Приложения

Приложение 1

Характеристика таблиц БД «Учебный процесс»

1. Банк вакансий

Имя поля	Тип данных	Знаков	Описание
кпрт	числовой		Код предприятия
ваканс	текстовый	50	Вакансии для студентов

2. Вид приказа

Имя поля	Тип данных	Знаков	Описание
кприк	числовой		Код приказа
вид приказа	текстовый	50	Об отчислении, отпуске, восстановлении
назвпр	текстовый	50	Название приказа
датапр	дата/время		Дата приказа

3. Группа

Имя поля	Тип данных	Знаков	Описание
нг	текстовый	6	N группы
название	текстовый	6	Об отчислении, отпуске, восстановлении
отд	текстовый	20	Отделение
староста	текстовый	20	
старполн	текстовый	50	Полные фамилия, имя, отчество старосты

Продолжение табл.

Имя поля	Тип данных	Знаков	Описание
куратор	текстовый	20	
курс	текстовый	6	
шспец	числовой		
год набора	числовой		
студентов	числовой		
девушек	числовой		
юношей	числовой		
бесплат	числовой		Бесплатная форма
частич	числовой		Частичноплатная форма
полнопл	числовой		Полноплатная форма

4. Изучение

Имя поля	Тип данных	Знаков	Описание
нг	текстовый	6	
кпред	числовой		Код предприятия
учплан	числовой		Учебный план
семестр	числовой		
кпреп	числовой		Код преподавателя
вид занятий	текстовый	20	Лекции, лабораторные, практические занятия, курсовые работы
часы	числовой		
отчетн	текстовый	10	Экзамен, зачет, курсовая работа, аттестация

5. Кафедра

Имя поля	Тип данных	Знаков	Описание
ккаф	числовой		Код кафедры
назвкаф	текстовый	60	Название кафедры

Продолжение табл.

Имя поля	Тип данных	Знаков	Описание
аббрев	текстовый	6	Аббревиатура кафедры
заведующ	текстовый	30	
учстеп	текстовый	10	Ученая степень
учзван	текстовый	10	Ученое звание
рабтел	числовой		Рабочий телефон
домтел	текстовый	10	Домашний телефон
замест	текстовый	30	Зам. зав. кафедрой
секретарь	текстовый	30	Секретарь кафедры

6. Лаборатория

Имя поля	Тип данных	Знаков	Описание
ккаф	числовой		
лаборатория	текстовый	50	

7. Общежитие

Имя поля	Тип данных	Знаков	Описание
нзк	текстовый	10	
семестр	числовой		
догобщ	числовой		Договор об общежитии
адробщ	текстовый	50	Адрес общежития

8. Оплата

Имя поля	Тип данных	Знаков	Описание
нг	текстовый	6	
нзк	текстовый	10	
платдок	числовой		Платежный документ
дата	числовой		Дата оплаты
сум	числовой		Сумма оплаты

9. Предмет

Имя поля	Тип данных	Знаков	Описание
кпред	числовой		Код предмета
учплан	числовой		Учебный план (год)
предмет	текстовый	40	Название дисциплины
лекции	числовой		
лабр	числовой		Лабораторные работы
прраб	числовой		Практические работы
конраб	числовой		Контрольные работы
курсп	числовой		Курсовой проект (работа)
часы общ	числовой		
часы ауд	числовой		
ксем	числовой		Количество семестров

10. Предприятие

Имя поля	Тип данных	Знаков	Описание
кпр	числовой		Код предприятия
назвипп	текстовый	50	Название ИПП
адрипп	текстовый	50	Адрес предприятия ИПП
проф	текстовый	30	Профиль предприятия
телипп	текстовый	15	Телефон предприятия
расчет	числовой		Расчетный счет

11. Преподаватель

Имя поля	Тип данных	Знаков	Описание
кпреп	числовой		Табельный номер преподавателя
ккаф	числовой		
фиопреп	текстовый	40	Фамилия, имя, отчество преподавателя
должпреп	текстовый	20	Должность преподавателя
фамио	текстовый	20	Фамилия и инициалы
статус	текстовый	10	Штатный/нештатный
учстеп	текстовый	10	Ученая степень
учзван	текстовый	10	Ученое звание
домтел	числовой		Домашний телефон

12. Приказы на студента

Имя поля	Тип данных	Знаков	Описание
нзк	текстовый	6	
семестр	числовой		
кприк	числовой		Код приказа
причина	текстовый	20	

13. Специальность

Имя поля	Тип данных	Знаков	Описание
ккаф	числовой		
шспец	числовой		
назвспец	текстовый	70	Название специальности
срокобуч	текстовый	15	Срок обучения

14. Студент

Имя поля	Тип данных	Знаков	Описание
нг	текстовый	6	
нзк	текстовый	10	
номерстуд	числовой		Номер студента в группе
фамилия и о	текстовый	50	Фамилия, имя, отчество студента
старфам	текстовый	20	Старая фамилия
стипендия	текстовый	10	
датарож	дата/время		Дата рождения
паспорт	текстовый	15	Данные паспорта
адрес	текстовый	50	Адрес проживания в городе
горприб	текстовый	20	Город прибытия на учебу
адрприб	текстовый	50	Адрес прибытия
вид оплаты	текстовый	10	
отец	текстовый	50	
мать	текстовый	50	
военнооб	логический		Военнообязанный
годпос	числовой		Год поступления
пол	текстовый	5	
нкред	числовой		Номер кредитной карточки
образ	текстовый	20	Образование (школа, техникум, лицей)
спол	текстовый	20	Семейное положение
ижд	текстовый	20	Наличие иждивенцев
осотм	текстовый	20	Особые отметки (сирота и т. д.)

Продолжение табл.

Имя поля	Тип данных	Знаков	Описание
иняз	текстовый	15	Знание иностранного языка
водит	логический		Наличие водительских прав
домтел	текстовый	12	Домашний телефон
мработы	текстовый	30	Место работы студента
должс	текстовый	40	Должность, на которой работает студент

15. Успеваемость

Имя поля	Тип данных	Знаков	Описание
нг	текстовый	6	
нзк	текстовый	10	
кпред	числовой		
учплан	числовой		
семестр	числовой		
кпреп	числовой		
отчетн	текстовый	10	
оценка	текстовый	10	Зачет, отлично, хорошо, удовлетворительно, неявка, незачет
после сессии	логический		Пометка о сдаче отчетности после сессии
отметка	числовой		3, 4, 5 для дальнейших числовых расчетов

Приложение 2

Программа серверной части РБД

```
/* Extract Database E:\Diplom\PROJECT\dekanat\Bases.gdb */
CREATE DATABASE «E:\Diplom\PROJECT\dekanat\Bases.gdb»
PAGE_SIZE 1024
DEFAULT CHARACTER SET WIN1251;
```

```
/* Table: ARHIVSTUD, Owner: SYSDBA */
CREATE TABLE ARHIVSTUD (NG VARCHAR(6) CHARACTER SET
WIN1251,
    NZK VARCHAR(10) CHARACTER SET WIN1251 NOT NULL,
    FIOPREP VARCHAR(40) CHARACTER SET WIN1251,
    UCH_PLAN INTEGER,
    SEMESTR INTEGER,
    PREDM VARCHAR(60) CHARACTER SET WIN1251,
    OTCHET VARCHAR(10) CHARACTER SET WIN1251,
    OCENKA VARCHAR(20) CHARACTER SET WIN1251,
    OTMETKA INTEGER);

/* Table: ATESTACIA, Owner: SYSDBA */
CREATE TABLE ATESTACIA (NG VARCHAR(6) CHARACTER SET
WIN1251 NOT NULL,
    NZK VARCHAR(10) CHARACTER SET WIN1251 NOT NULL,
    KPRED INTEGER NOT NULL,
    UCH_PLAN INTEGER,
    SEMESTR INTEGER,
    KPREP INTEGER NOT NULL,
    OTCHETN VARCHAR(10) CHARACTER SET WIN1251,
    OCENKA VARCHAR(20) CHARACTER SET WIN1251,
    OTMETKA INTEGER DEFAULT 1
);

/* Table: BANK_VAK, Owner: SYSDBA */
CREATE TABLE BANK_VAK (KPRP INTEGER NOT NULL,
    VAKANS VARCHAR(50) CHARACTER SET WIN1251);

/* Table: GRUPPA, Owner: SYSDBA */
CREATE TABLE GRUPPA (NG VARCHAR(6) CHARACTER SET
WIN1251 NOT NULL,
    NAZVANIE VARCHAR(6) CHARACTER SET WIN1251,
    OTD VARCHAR(20) CHARACTER SET WIN1251,
    STAROSTA VARCHAR(20) CHARACTER SET WIN1251,
    STAR_POLN VARCHAR(50) CHARACTER SET WIN1251,
    KURATOR VARCHAR(20) CHARACTER SET WIN1251,
    KURS VARCHAR(50) CHARACTER SET WIN1251,
    SH_SPEC INTEGER NOT NULL,
    GOD_NABOR INTEGER,
    STUDENTOV INTEGER,
    DEVUSHEK INTEGER,
    UNOSHEY INTEGER,
    BESPLATN INTEGER,
```

```
    CHAST INTEGER,
    POLNPLATN INTEGER,
CONSTRAINT GRUPPRIMKEY1 PRIMARY KEY (NG));

/* Table: IPP, Owner: SYSDBA */
CREATE TABLE IPP (KPRP INTEGER NOT NULL,
    NZK VARCHAR(10) CHARACTER SET WIN1251 NOT NULL,
    SEMESTR INTEGER,
    VR_PRAKT DATE,
    DOLGN_IPP VARCHAR(50) CHARACTER SET WIN1251);

/* Table: IZUCHENIE, Owner: SYSDBA */
CREATE TABLE IZUCHENIE (NG VARCHAR(10) CHARACTER SET
WIN1251 NOT NULL,
    KPRED INTEGER NOT NULL,
    UCH_PLAN INTEGER NOT NULL,
    SEMESTR INTEGER NOT NULL,
    KPREP INTEGER NOT NULL,
    VIDZANATIA VARCHAR(50) CHARACTER SET WIN1251,
    CHASI INTEGER,
    OTCHET VARCHAR(10) CHARACTER SET WIN1251);

/* Table: KAFEDRA, Owner: SYSDBA */
CREATE TABLE KAFEDRA (KKAF INTEGER NOT NULL,
    NAZV_KAF VARCHAR(60) CHARACTER SET WIN1251,
    ABREV VARCHAR(6) CHARACTER SET WIN1251,
    ZAVED VARCHAR(30) CHARACTER SET WIN1251,
    UCH_STEP VARCHAR(10) CHARACTER SET WIN1251,
    UCH_ZVAN VARCHAR(10) CHARACTER SET WIN1251,
    RABTEL VARCHAR(14) CHARACTER SET WIN1251,
    DOMTEL VARCHAR(14) CHARACTER SET WIN1251,
    ZAMEST VARCHAR(30) CHARACTER SET WIN1251,
    SEKRETAR VARCHAR(30) CHARACTER SET WIN1251,
CONSTRAINT KAFEDRAPKEY1 PRIMARY KEY (KKAF));

/* Table: LABA, Owner: SYSDBA */
CREATE TABLE LABA (KKAF INTEGER NOT NULL,
    LABORATOR VARCHAR(50) CHARACTER SET WIN1251);

/* Table: OBSHEGIT, Owner: SYSDBA */
CREATE TABLE OBSHEGIT (NZK VARCHAR(10) CHARACTER SET
WIN1251 NOT NULL,
    SEMESTR INTEGER,
    DOG_OBSH INTEGER,
    ADRES_OBSH VARCHAR(50) CHARACTER SET WIN1251);
```

```
/* Table: OPLATA, Owner: SYSDBA */
CREATE TABLE OPLATA (NG VARCHAR(6) CHARACTER SET
WIN1251 NOT NULL,
    NZK VARCHAR(10) CHARACTER SET WIN1251 NOT NULL,
    PLATDOK INTEGER DEFAULT 0

    DATAD DATE,
    SUMMA FLOAT);

/* Table: PREDMET, Owner: SYSDBA */
CREATE TABLE PREDMET (KPRED INTEGER NOT NULL,
    UCH_PLAN INTEGER NOT NULL,
    PREDM VARCHAR(60) CHARACTER SET WIN1251,
    SOKRPRED VARCHAR(20) CHARACTER SET WIN1251,
    LEKCII INTEGER,
    LABR INTEGER,
    PRRAB INTEGER,
    KRRAB INTEGER,
    KURSP INTEGER,
    CHAS_OBCH INTEGER,
    CHAS_AUDIT INTEGER,
    K_SEMESTR INTEGER,
CONSTRAINT PREDMETPKEY1 PRIMARY KEY (KPRED));

/* Table: PREDPR, Owner: SYSDBA */
CREATE TABLE PREDPR (KPRP INTEGER NOT NULL,
    NAZVPRP VARCHAR(50) CHARACTER SET WIN1251,
    ADRPRP VARCHAR(50) CHARACTER SET WIN1251,
    PROFIL VARCHAR(30) CHARACTER SET WIN1251,
    TELPRP VARCHAR(30) CHARACTER SET WIN1251,
    RASCHET VARCHAR(50) CHARACTER SET WIN1251,
    DOGOVOR INTEGER,
CONSTRAINT PREDPRPKEY1 PRIMARY KEY (KPRP));

/* Table: PREPOD, Owner: SYSDBA */
CREATE TABLE PREPOD (KPREP INTEGER NOT NULL,
    KKAF INTEGER NOT NULL,
    FIOPREP VARCHAR(40) CHARACTER SET WIN1251,
    DOLG_PREP VARCHAR(20) CHARACTER SET WIN1251,
    FAMIO VARCHAR(20) CHARACTER SET WIN1251,
    STATUS VARCHAR(50) CHARACTER SET WIN1251,
    UCH_STEP VARCHAR(10) CHARACTER SET WIN1251,
    UCH_ZVAN VARCHAR(10) CHARACTER SET WIN1251,
    DOMTEL VARCHAR(14) CHARACTER SET WIN1251,
CONSTRAINT PREPODPKEY1 PRIMARY KEY (KPREP));
```

```
/* Table: PRIKAZ, Owner: SYSDBA */
CREATE TABLE PRIKAZ (NZK VARCHAR(10) CHARACTER SET
WIN1251,
    SEMESTR INTEGER,
    KPRIKAZ INTEGER NOT NULL,
    PRICHINA VARCHAR(20) CHARACTER SET WIN1251,
CONSTRAINT PRIKAZPKEY1 PRIMARY KEY (KPRIKAZ));

/* Table: SPECIALNOST, Owner: SYSDBA */
CREATE TABLE SPECIALNOST (KKAF INTEGER NOT NULL,
    SH_SPEC INTEGER NOT NULL,
    NAZVSPEC VARCHAR(70) CHARACTER SET WIN1251,
    SROK_OBUCH VARCHAR(15) CHARACTER SET WIN1251,
CONSTRAINT SPECIALNOSTPRKEY1 PRIMARY KEY (SH_SPEC));

/* Table: STUDENT, Owner: SYSDBA */
CREATE TABLE STUDENT (NZK VARCHAR(10) CHARACTER SET
WIN1251 NOT NULL,
    NG VARCHAR(6) CHARACTER SET WIN1251,
    NOMER_STUD INTEGER default 0
,
    FIO VARCHAR(50) CHARACTER SET WIN1251,
    STARFAM VARCHAR(20) CHARACTER SET WIN1251,
    STIPENDIA VARCHAR(10) CHARACTER SET WIN1251,
    DATAROJ DATE,
    PASPORT VARCHAR(15) CHARACTER SET WIN1251,
    ADRES VARCHAR(50) CHARACTER SET WIN1251,
    GORPRIB VARCHAR(20) CHARACTER SET WIN1251,
    ADRPRIB VARCHAR(50) CHARACTER SET WIN1251,
    VID_OPLAT VARCHAR(10) CHARACTER SET WIN1251,
    OTEC VARCHAR(50) CHARACTER SET WIN1251,
    MAT VARCHAR(50) CHARACTER SET WIN1251,
    VOEN_OB CHAR(1) CHARACTER SET WIN1251,
    GOD_POST INTEGER,
    POL VARCHAR(5) CHARACTER SET WIN1251,
    NKRED INTEGER,
    OBRAZ VARCHAR(20) CHARACTER SET WIN1251,
    S_POL VARCHAR(20) CHARACTER SET WIN1251,
    IJD VARCHAR(20) CHARACTER SET WIN1251,
    OS_OTM VARCHAR(20) CHARACTER SET WIN1251,
    INYAZ VARCHAR(15) CHARACTER SET WIN1251,
    VODIT CHAR(1) CHARACTER SET WIN1251,
    DOMTEL VARCHAR(12) CHARACTER SET WIN1251,
    MRABOT VARCHAR(30) CHARACTER SET WIN1251,
    DOLGN VARCHAR(50) CHARACTER SET WIN1251,
    AKADEM_OTP CHAR(1) CHARACTER SET WIN1251,
PRIMARY KEY (NZK));
```

```
/* Table: TEMPUSP, Owner: SYSDBA */
CREATE TABLE TEMPUSP (NZK VARCHAR(10) CHARACTER SET
WIN1251 NOT NULL,
    OTCHET VARCHAR(10) CHARACTER SET WIN1251,
    SEMESTR INTEGER NOT NULL,
    SH_SPEC INTEGER NOT NULL,
    POSL_SES CHAR(1) CHARACTER SET WIN1251,
    FIO VARCHAR(50) CHARACTER SET WIN1251,
    KURS INTEGER,
    OCENKA VARCHAR(20) CHARACTER SET WIN1251);

/* Table: USPEVAEM, Owner: SYSDBA */
CREATE TABLE USPEVAEM (NG VARCHAR(6) CHARACTER SET
WIN1251 NOT NULL,
    NZK VARCHAR(10) CHARACTER SET WIN1251 NOT NULL,
    KPRED INTEGER NOT NULL,
    UCH_PLAN INTEGER,
    SEMESTR INTEGER,
    KPREP INTEGER NOT NULL,
    OTCHET VARCHAR(10) CHARACTER SET WIN1251,
    OCENKA VARCHAR(20) CHARACTER SET WIN1251,
    POSL_SES CHAR(1) CHARACTER SET WIN1251,
    OTMETKA INTEGER);

/* Table: VID_PRIKAZ, Owner: SYSDBA */
CREATE TABLE VID_PRIKAZ (KPRIK INTEGER NOT NULL,
    VID_PRIK VARCHAR(50) CHARACTER SET WIN1251,
    NAZV_PRIK VARCHAR(50) CHARACTER SET WIN1251,
    DATA_PRIK DATE);
ALTER TABLE STUDENT ADD CONSTRAINT STUDENTFORKEY1
FOREIGN KEY (NG) REFERENCES GRUPPA(NG);
ALTER TABLE OPLATA ADD CONSTRAINT OPLATAFORKEY1
FOREIGN KEY (NZK) REFERENCES STUDENT(NZK);
ALTER TABLE OPLATA ADD CONSTRAINT OPLATAFORKEY2
FOREIGN KEY (NG) REFERENCES GRUPPA(NG);
ALTER TABLE SPECIALNOST ADD CONSTRAINT
SPECIALNOSTFKEY1 FOREIGN KEY (KKAF) REFERENCES
KAFEDRA(KKAF);
ALTER TABLE PREPOD ADD CONSTRAINT PREPODFKEY1
FOREIGN KEY (KKAF) REFERENCES KAFEDRA(KKAF);
ALTER TABLE IZUCHENIE ADD CONSTRAINT IZUCHENIEFKEY1
FOREIGN KEY (NG) REFERENCES GRUPPA(NG);
ALTER TABLE IZUCHENIE ADD CONSTRAINT IZUCHENIEFKEY2
FOREIGN KEY (KPREP) REFERENCES PREPOD(KPREP);
ALTER TABLE USPEVAEM ADD CONSTRAINT USPEVAEMFKEY2
FOREIGN KEY (NZK) REFERENCES STUDENT(NZK);
ALTER TABLE IZUCHENIE ADD CONSTRAINT IZUCHENIEFKEY3
FOREIGN KEY (KPRED) REFERENCES PREDMET(KPRED);
```

```
    ALTER TABLE USPEVAEM ADD CONSTRAINT USPEVAEMFKEY3
FOREIGN KEY (KPRED) REFERENCES PREDMET(KPRED);
    ALTER TABLE USPEVAEM ADD CONSTRAINT USPEVAEMFKEY4
FOREIGN KEY (KPREP) REFERENCES PREPOD(KPREP);
    ALTER TABLE PRIKAZ ADD CONSTRAINT PRIKAZFKEY1 FOREIGN
KEY (NZK) REFERENCES STUDENT(NZK);
    ALTER TABLE USPEVAEM ADD CONSTRAINT USPEVAEMFKEY1
FOREIGN KEY (NG) REFERENCES GRUPPA(NG);
    ALTER TABLE VID_PRIKAZ ADD CONSTRAINT VID_PRIKAZFKEY1
FOREIGN KEY (KPRIK) REFERENCES PRIKAZ(KPRIKAZ);
    ALTER TABLE ATESTACIA ADD CONSTRAINT ATESTACIAFKEY1
FOREIGN KEY (NG) REFERENCES GRUPPA(NG);
    ALTER TABLE ATESTACIA ADD CONSTRAINT ATESTACIAFKEY2
FOREIGN KEY (NZK) REFERENCES STUDENT(NZK);
    ALTER TABLE ATESTACIA ADD CONSTRAINT ATESTACIAFKEY3
FOREIGN KEY (KPRED) REFERENCES PREDMET(KPRED);
    ALTER TABLE ATESTACIA ADD CONSTRAINT ATESTACIAFKEY4
FOREIGN KEY (KPREP) REFERENCES PREPOD(KPREP);
    ALTER TABLE OBSHEGIT ADD CONSTRAINT OBSHEGITFKEY1
FOREIGN KEY (NZK) REFERENCES STUDENT(NZK);
    ALTER TABLE BANK_VAK ADD CONSTRAINT BANK_VAKFKEY1
FOREIGN KEY (KPRP) REFERENCES PREDPR(KPRP);
    ALTER TABLE IPP ADD CONSTRAINT IPPFKEY1 FOREIGN KEY
(KPRP) REFERENCES PREDPR(KPRP);
    ALTER TABLE IPP ADD CONSTRAINT IPPFKEY2 FOREIGN KEY
(NZK) REFERENCES STUDENT(NZK);
    ALTER TABLE LABA ADD CONSTRAINT LABAFKEY1 FOREIGN KEY
(KKAF) REFERENCES KAFEDRA(KKAF);
    ALTER TABLE GRUPPA ADD CONSTRAINT GRUPPAFKEY1
FOREIGN KEY (SH_SPEC) REFERENCES SPECIALNOST(SH_SPEC);
    ALTER TABLE ARHIVSTUD ADD CONSTRAINT ARHIVSTUDFKEY
FOREIGN KEY (NZK) REFERENCES STUDENT(NZK);

    CREATE GENERATOR GEN_KPRP;
    CREATE GENERATOR GEN_KKAF;
    CREATE GENERATOR GEN_KPRED;

    /* View: EKZAMVIEW, Owner: SYSDBA */
    CREATE VIEW EKZAMVIEW (NZK, OTCHET, SEMESTR, SH_SPEC,
POSL_SES, FIO, KURS) AS

    SELECT uspevaem.nzk,otchet,uspevaem.semestr,gruppa.sh_spec,posl_ses,
student.fio,gruppa.kurs
    from uspevaem, gruppa, student
    where uspevaem.ng=gruppa.ng and
    uspevaem.nzk=student.nzk and semestr <> 12
```

```
    group by uspevaem.nzk,otchet,uspevaem.semestr,sh_spec,posl_ses,student.fio,
gruppa.kurs
    having otchet='экз'

    ;

    /* View: GRUPPAVIEW, Owner: SYSDBA */
    CREATE VIEW GRUPPAVIEW (NG, NAZVANIE) AS
     SELECT ng,nazvanie from gruppa

    ;

    /* View: PREDSELECT, Owner: SYSDBA */
    CREATE VIEW PREDSELECT (KPRED, PREDM) AS
     SELECT kpred, predm from predmet

    ;

    /* View: STUDENTVIEW, Owner: SYSDBA */
    CREATE VIEW STUDENTVIEW (NZK) AS
     SELECT NZK from student

    ;

    /* View: ZADOLGVIEW, Owner: SYSDBA */
    CREATE VIEW ZADOLGVIEW (NZK, OTCHET, SEMESTR, KPRED,
POSL_SES, FIO, KURS) AS

    SELECT uspevaem.nzk,otchet,uspevaem.semestr,uspevaem.kpred,posl_ses,
student.fio,gruppa.kurs
    from uspevaem, gruppa, student
    where uspevaem.ng=gruppa.ng and
    uspevaem.nzk=student.nzk and semestr <> 12
    group by uspevaem.nzk,otchet,uspevaem.semestr,kpred,posl_ses,student.fio,
gruppa.kurs
    having otchet='экз'

    ;
    COMMIT WORK;
    SET AUTODDL OFF;
    SET TERM ^ ;

    /* Stored procedures */
    CREATE PROCEDURE GRUPPA_DEL AS BEGIN EXIT; END ^
    CREATE PROCEDURE STUDENT_DEL AS BEGIN EXIT; END ^
    CREATE PROCEDURE KAFEDRA_DEL AS BEGIN EXIT; END ^
```

```
CREATE PROCEDURE SPECIALNOST_DEL AS BEGIN EXIT; END ^
CREATE PROCEDURE PREDMET_DEL AS BEGIN EXIT; END ^
CREATE PROCEDURE PREPOD_DEL AS BEGIN EXIT; END ^
CREATE PROCEDURE PRIKAZ_DEL AS BEGIN EXIT; END ^
CREATE PROCEDURE PREDPR_DEL AS BEGIN EXIT; END ^
CREATE PROCEDURE GRUP_SUM AS BEGIN EXIT; END ^

ALTER PROCEDURE GRUPPA_DEL (DNG VARCHAR(6)
CHARACTER SET WIN1251)

AS
declare variable dnzk varchar(10);
declare variable dkprikaz integer;
BEGIN
for select nzk from student
where student.ng=:dng
into :dnzk
do
begin
    delete from ipp where ipp.nzk=:dnzk;
    delete from uspevaem where uspevaem.nzk=:dnzk;
    delete from atestacia where atestacia.nzk=:dnzk;
    delete from oplata where oplata.nzk=:dnzk;
    delete from obshegit where obshegit.nzk=:dnzk;
    delete from arhivstud where arhivstud.nzk=:dnzk;
        for select kprikaz from prikaz
        where prikaz.nzk=:dnzk
        into :dkprikaz
        do delete from vid_prikaz where vid_prikaz.kprik=:dkprikaz;

delete from prikaz where prikaz.nzk=:dnzk;
end
delete from student where student.ng=:dng;
delete from izuchenie where izuchenie.ng=:dng;
delete from gruppa where gruppa.ng=:dng;

suspend;
END
^

ALTER PROCEDURE STUDENT_DEL (DNZK VARCHAR(10)
CHARACTER SET WIN1251)
AS
declare variable dkprikaz integer;
BEGIN
```

```
        delete from ipp where ipp.nzk=:dnzk;
        delete from uspevaem where uspevaem.nzk=:dnzk;
        delete from atestacia where atestacia.nzk=:dnzk;
        delete from oplata where oplata.nzk=:dnzk;
        delete from obshegit where obshegit.nzk=:dnzk;
        delete from arhivstud where arhivstud.nzk=:dnzk;
            for select kprikaz from prikaz
            where prikaz.nzk=:dnzk
            into :dkprikaz
            do delete from vid_prikaz where vid_prikaz.kprik=:dkprikaz;

delete from prikaz where prikaz.nzk=:dnzk;
delete from student where student.nzk=:dnzk;
suspend;
END
^

ALTER PROCEDURE KAFEDRA_DEL (DKKAF INTEGER)
AS
declare variable dsh_spec integer;
declare variable dng varchar(6);
declare variable dnzk varchar(10);
declare variable dkprikaz integer;
declare variable dkprep integer;

BEGIN
for select sh_spec from specialnost
where kkaf=:dkkaf
into :dsh_spec
do
begin
    for select ng from gruppa
    where gruppa.sh_spec=:dsh_spec
    into :dng
    do
    begin
    for select nzk from student
    where student.ng=:dng
    into :dnzk
    do
    begin
        delete from ipp where ipp.nzk=:dnzk;
        delete from uspevaem where uspevaem.nzk=:dnzk;
        delete from atestacia where atestacia.nzk=:dnzk;
        delete from oplata where oplata.nzk=:dnzk;
        delete from obshegit where obshegit.nzk=:dnzk;
```

```
        delete from arhivstud where arhivstud.nzk=:dnzk;

            for select kprikaz from prikaz
            where prikaz.nzk=:dnzk
            into :dkprikaz
            do delete from vid_prikaz where vid_prikaz.kprik=:dkprikaz;

delete from prikaz where prikaz.nzk=:dnzk;
end
delete from student where student.ng=:dng;
    delete from izuchenie where izuchenie.ng=:dng;
    delete from gruppa where gruppa.ng=:dng;
end
    delete from specialnost where sh_spec=:dsh_spec;
end

for select kprep from prepod where kkaf=:dkkaf
into :dkprep
do begin
    delete from izuchenie where kprep=:dkprep;
    delete from uspevaem where kprep=:dkprep;
    delete from atestacia where kprep=:dkprep;
    end
delete from prepod where kkaf=:dkkaf;
delete from laba where laba.kkaf=:dkkaf;
delete from kafedra where kkaf=:dkkaf;
suspend;
END
^

ALTER PROCEDURE SPECIALNOST_DEL (DSH_SPEC INTEGER)
AS
declare variable dng varchar(6);
declare variable dnzk varchar(10);
declare variable dkprikaz integer;
declare variable dkprep integer;
BEGIN
    for select ng from gruppa
    where gruppa.sh_spec=:dsh_spec
    into :dng
    do
    begin
    for select nzk from student
    where student.ng=:dng
    into :dnzk
    do
    begin
```

```
        delete from ipp where ipp.nzk=:dnzk;
        delete from uspevaem where uspevaem.nzk=:dnzk;
        delete from atestacia where atestacia.nzk=:dnzk;
        delete from oplata where oplata.nzk=:dnzk;
        delete from obshegit where obshegit.nzk=:dnzk;
        delete from arhivstud where arhivstud.nzk=:dnzk;

            for select kprikaz from prikaz
            where prikaz.nzk=:dnzk
            into :dkprikaz
            do delete from vid_prikaz where vid_prikaz.kprik=:dkprikaz;

    delete from prikaz where prikaz.nzk=:dnzk;
    end
    delete from student where student.ng=:dng;
    delete from izuchenie where izuchenie.ng=:dng;
    delete from gruppa where gruppa.ng=:dng;
end
    delete from specialnost where sh_spec=:dsh_spec;
SUSPEND;
END
^

ALTER PROCEDURE PREDMET_DEL (DKPRED INTEGER)
AS
BEGIN
delete from izuchenie where kpred=:dkpred;
delete from uspevaem where kpred=:dkpred;
delete from atestacia where kpred=:dkpred;
delete from predmet where kpred=:dkpred;
SUSPEND;
END
^

ALTER PROCEDURE PREPOD_DEL (DKPREP INTEGER)
AS
BEGIN
delete from izuchenie where kprep=:dkprep;
delete from uspevaem where kprep=:dkprep;
delete from atestacia where kprep=:dkprep;
delete from prepod where kprep=:dkprep;
SUSPEND;
END
^

ALTER PROCEDURE PRIKAZ_DEL (DKPRIKAZ INTEGER)
AS
```

```
    BEGIN
delete from vid_prikaz where kprik=:dkprikaz;
delete from prikaz where kprikaz=:dkprikaz;
SUSPEND;
END
^

ALTER PROCEDURE PREDPR_DEL (DKPRP INTEGER)
AS
    BEGIN
delete from ipp where kprp=:dkprp;
delete from bank_vak where kprp=:dkprp;
delete from predpr where kprp=:dkprp;
SUSPEND;
END
^

ALTER PROCEDURE GRUP_SUM AS

declare variable st1 integer;
declare variable st2 integer;
declare variable st3 integer;
declare variable st4 integer;
declare variable st5 integer;
declare variable st6 integer;
declare variable ngsum varchar(6);
BEGIN
for select ng from gruppa
into :ngsum
do
    begin
    st1=0;
    st2=0;
    st3=0;
    st4=0;
    st5=0;
    st6=0;
    select count (nzk) from student where student.ng=:ngsum into :st1;
    select count (nzk) from student where (student.ng=:ngsum)
    and (student.pol collate pxw_cyrl='ж') into :st2;
    select count (nzk) from student where (student.ng=:ngsum)
    and (student.pol collate pxw_cyrl='м') into :st3;
    select count (nzk) from student where (student.ng=:ngsum)
```

```
    and (student.vid_oplat collate pxw_cyrl='беспл') into :st4;
    select count (nzk) from student where (student.ng=:ngsum)
    and (student.vid_oplat collate pxw_cyrl='частич') into :st5;
    select count (nzk) from student where (student.ng=:ngsum)
    and (student.vid_oplat collate pxw_cyrl='полнопл') into :st6;
    update gruppa
    set studentov=:st1, devushek=:st2, UNOSHEY=:ST3, besplatn=:st4,
    chast=:st5, polnplatn=:st6
    where ng=:ngsum;
    end
suspend;
END
^
SET TERM ; ^
COMMIT WORK ;
SET AUTODDL ON;
SET TERM ^ ;

/* Triggers only will work for SQL triggers */
CREATE TRIGGER KPRP_NO_GEN FOR PREDPR
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
predpr.kprp=gen_id(gen_kprp,1);
END
^
CREATE TRIGGER KPRED_NO_GEN FOR PREDMET
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
predmet.kpred=gen_id(gen_kpred,1);
END
^
CREATE TRIGGER KKAF_NO_GEN FOR KAFEDRA
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
kafedra.kkaf=gen_id(gen_kkaf,1);
END
^
COMMIT WORK ^
SET TERM ; ^

/* Grant permissions for this database */
```

Алгоритм приложения

В этом приложении отображены серверная и клиентская части программы режима клиент—сервер (с использованием хранимых процедур). Приведем фрагмент программы.

Серверная часть

```
/* Extract Database D:\students\Chert_cl\Priems.gdb */
CREATE DATABASE 'D:\students\Chert_cl\Priems.gdb' PAGE_SIZE 1024 ;

/* Table: KADRY, Owner: SYSDBA */
CREATE TABLE KADRY (VREM INTEGER,
    DOLGNOST CHAR(20),
    FIO CHAR(15),
    STEPEN CHAR(10),
    OTKRITIE CHAR(5),
    AVG_BAL DOUBLE DESISION,
    STAG INTEGER,
    STATUS CHAR(5),
    NBR_RULE INTEGER);

/* Table: RULE, Owner: SYSDBA */
CREATE TABLE RULE (NBR_RULE INTEGER,
    STEPEN CHAR(5),
    OTKRITIE CHAR(5),
    ZNAK_BAL CHAR(5),
    AVG_BAL DOUBLE DECISION,
    ZNAK_STAZ CHAR(5),
    STAG INTEGER,
    DOLGNOST CHAR(20),
    INFORM CHAR(255));

/* Table: STAT, Owner: SYSDBA */
CREATE TABLE STAT (VREM INTEGER,
    DOLGN CHAR(20),
    PLAN_R INTEGER,
    FACT_R INTEGER,
    VAKANS INTEGER,
    PRINIM INTEGER);

COMMIT WORK;
SET AUTODDL OFF;
SET TERM ^ ;
```

```
/* Stored procedures */
CREATE PROCEDURE PRO_KADRY AS BEGIN EXIT; END ^
ALTER PROCEDURE PRO_KADRY (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER, OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15), OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5), OUT_AVG_BAL DOUBLE DESISION,
OUT_STAG INTEGER, OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS
    BEGIN
    FOR SELECT VREM, DOLGNOST, FIO, STEPEN, OTKRITIE,
AVG_BAL, STAG, STATUS, NBR_RULE
    FROM KADRY
    WHERE VREM = :IN_VREM
    INTO :
    OUT_VREM, :OUT_DOLGNOST, :OUT_FIO, OUT_STEPEN,
:OUT_OTKRITIE, :OUT_AVG_BAL, : OUT_STAG, :OUT_STATUS,
:OUT_NBR_RULE
DO
    SUSPEND;
    END
^
SET TERM ; ^
COMMIT WORK ;
SET AUTODDL ON;

/* Grant permissions for this database */
```

Клиентская часть

В компонент Query1 (свойство SQL) необходимо записать

```
select * from pro_kadry (:param1)
```

а в файл unit1.pas добавить

```
procedure TForm1.N18Click(Sender: TObject);
begin
with Query1 do
begin
    Close;
ParamByName('param1').Value:=StrToInt(Edit1.Text);
    Open;
end;
end;
```

В приложениях 3.1 и 3.2 приведен полный текст программ БД «Прием на работу» для случая, когда каждая таблица с данными выводится на отдельную форму Delphi, а на первой форме Delphi расположены поля циклов времени, меню и объекты StoredProc.

В приложении 3.1 помещена программа на вложенном языке SQL, реализованная на сервере. Она включает процесс создания БД, таблиц и хранимые процедуры.

В приложении 3.2 размещены клиентская программа для локального варианта полного режима клиент—сервер. В Unit1 размещены программы вызова модальных форм. В остальных Uniti — процедуры, выполняемые при нажатии соответствующих элементов меню и кнопок. Обновление данных в БД проводится двумя способами: через компонент StoredProc и непосредственно в программе на языке Object Pascal. Вызов данных (select) проводится через компонент Query в виде запроса с параметрами, при этом оператор select размещается либо в свойстве SQL, либо в программе на языке Object Pascal (формируемый запрос на встроенном языке SQL). В первом случае вид select-оператора приведен в скобках-комментариях {}.

ПРИЛОЖЕНИЕ 3.1

Программа на сервере

```
/* Extract Database C:\Documents and Settings\i5.WORK\Рабочий
стол\Chert_cl\PRIEMS.GDB */
CREATE DATABASE 'C:\Documents and Settings\i5.WORK\Рабочий
стол\Chert_cl\PRIEMS.GDB' PAGE_SIZE 1024
;

/* Table: KADRY, Owner: SYSDBA */
CREATE TABLE KADRY (VREM INTEGER,
    DOLGNOST CHAR(20),
    FIO CHAR(15),
    STEPEN CHAR(10),
    OTKRITIE CHAR(5),
    STAG INTEGER,
    STATUS CHAR(5),
    NBR_RULE INTEGER,
    AVG_BAL NUMERIC(15, 2));

/* Table: KADRY_ISH, Owner: SYSDBA */
CREATE TABLE KADRY_ISH (VREM INTEGER,
    DOLGNOST CHAR(20),
    FIO CHAR(15),
    STEPEN CHAR(10),
    OTKRITIE CHAR(5),
    STAG INTEGER,
    STATUS CHAR(5),
```

```
    NBR_RULE INTEGER,
    AVG_BAL NUMERIC(15, 2));

/* Table: RULE, Owner: SYSDBA */
CREATE TABLE RULE (NBR_RULE INTEGER,
    STEPEN CHAR(5),
    OTKRITIE CHAR(5),
    ZNAK_BAL CHAR(5),
    ZNAK_STAZ CHAR(5),
    STAG INTEGER,
    DOLGNOST CHAR(20),
    INFORM CHAR(255),
    AVG_BAL NUMERIC(15, 2));

/* Table: RULE_ISH, Owner: SYSDBA */
CREATE TABLE RULE_ISH (NBR_RULE INTEGER,
    STEPEN CHAR(5),
    OTKRITIE CHAR(5),
    ZNAK_BAL CHAR(5),
    ZNAK_STAZ CHAR(5),
    STAG INTEGER,
    DOLGNOST CHAR(20),
    INFORM CHAR(255),
    AVG_BAL NUMERIC(15, 2));

/* Table: STAT, Owner: SYSDBA */
CREATE TABLE STAT (VREM INTEGER,
    DOLGN CHAR(20),
    PLAN_R INTEGER,
    FACT_R INTEGER,
    VAKANS INTEGER,
    PRINIM INTEGER,
    NEDOBOR INTEGER);

/* Table: STAT_ISH, Owner: SYSDBA */
CREATE TABLE STAT_ISH (VREM INTEGER,
    DOLGN CHAR(20),
    PLAN_R INTEGER,
    FACT_R INTEGER,
    VAKANS INTEGER,
    PRINIM INTEGER,
    NEDOBOR INTEGER);

COMMIT WORK;
SET AUTODDL OFF;
SET TERM ^ ;
```

```
/* Stored procedures */
CREATE PROCEDURE PRO_KADRY AS BEGIN EXIT; END ^
CREATE PROCEDURE PRP_ZAPRULE AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_KOLRES AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_RABOT AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_PRETE AS BEGIN EXIT; END ^
CREATE PROCEDURE NACHAT AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_STAT AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_PRNJAT AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_PRNIM AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_PROD1 AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_OBJAS AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_IZMPRAV AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_IZMRES AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_PRINWS AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_PROD2 AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_KADRYRES AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_STATRES AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_IZMRES2 AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_RABOT1 AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_UVOLI AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_RABOT2 AS BEGIN EXIT; END ^
CREATE PROCEDURE PRO_STATMIN AS BEGIN EXIT; END ^

ALTER PROCEDURE PRO_KADRY (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
        AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
```

```
    WHERE VREM BETWEEN 1 AND :IN_VREM
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    :OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRP_ZAPRULE (IN_VREM INTEGER)
AS

DECLARE VARIABLE DSTEPEN CHAR(10);
DECLARE VARIABLE DDOLGNOST CHAR(20);
DECLARE VARIABLE DOTKRITIE CHAR(5);
DECLARE VARIABLE DSTAG INTEGER;
DECLARE VARIABLE DAVG_BAL NUMERIC(15, 2);
DECLARE VARIABLE OLD_STATUS CHAR(5);
BEGIN
/* Pravilo1*/
    FOR SELECT
        STEPEN,
        DOLGNOST
            FROM RULE
                WHERE NBR_RULE=1
INTO :DSTEPEN,:DDOLGNOST

    DO
        BEGIN
            UPDATE KADRY
            SET NBR_RULE=1,
                DOLGNOST=:DDOLGNOST
                    WHERE STATUS='прете'
                        AND STEPEN=:DSTEPEN

                        AND VREM=:IN_VREM;
                    END

BEGIN
/* Pravilo2*/
    FOR SELECT
```

```
    STEPEN,
    OTKRITIE,

    DOLGNOST
        FROM RULE
            WHERE NBR_RULE=2
INTO :DSTEPEN, :DOTKRITIE,:DDOLGNOST

    DO
        BEGIN
            UPDATE KADRY
            SET NBR_RULE=2,
                DOLGNOST=:DDOLGNOST
                    WHERE STATUS='пратеж'
                        AND STEPEN=:DSTEPEN
                        AND OTKRITIE=:DOTKRITIE

                        AND VREM=:IN_VREM;
                END
END

BEGIN
/* Pravilo3*/
    FOR SELECT
        STEPEN,
        OTKRITIE,

        AVG_BAL,
        DOLGNOST
            FROM RULE
                WHERE NBR_RULE=3
INTO :DSTEPEN, :DOTKRITIE, :DAVG_BAL, :DDOLGNOST

    DO
        BEGIN
            UPDATE KADRY
            SET NBR_RULE=3,
                DOLGNOST=:DDOLGNOST
                    WHERE STATUS='пратеж'
                        AND STEPEN=:DSTEPEN
                        AND OTKRITIE=:DOTKRITIE

                        AND AVG_BAL>=:DAVG_BAL
                        AND VREM=:IN_VREM;
                END
END
```

```
    BEGIN
/* Pravilo4*/
    FOR SELECT
        STEPEN,
        OTKRITIE,
        STAG,
        AVG_BAL,
        DOLGNOST
            FROM RULE
                WHERE NBR_RULE=4
INTO :DSTEPEN, :DOTKRITIE, :DSTAG, :DAVG_BAL, :DDOLGNOST

    DO
        BEGIN
            UPDATE KADRY
            SET NBR_RULE=4,
                DOLGNOST=:DDOLGNOST
                    WHERE STATUS='прете'
                        AND STEPEN=:DSTEPEN
                        AND OTKRITIE=:DOTKRITIE
                        AND STAG>=:DSTAG
                        AND AVG_BAL<:DAVG_BAL
                        AND VREM=:IN_VREM;
                END
END

BEGIN
/* Pravilo5*/
    FOR SELECT
        STEPEN,
        OTKRITIE,
        STAG,
        AVG_BAL,
        DOLGNOST
            FROM RULE
                WHERE NBR_RULE=5
INTO :DSTEPEN, :DOTKRITIE, :DSTAG, :DAVG_BAL,
:DDOLGNOST DO
    BEGIN
        UPDATE KADRY
            SET NBR_RULE=5,
                DOLGNOST=:DDOLGNOST
                    WHERE STATUS='прете'
                        AND STEPEN=:DSTEPEN
                        AND OTKRITIE=:DOTKRITIE
                        AND STAG<:DSTAG
                        AND AVG_BAL<:DAVG_BAL
```

```
                    AND VREM=:IN_VREM;
            END
END

    BEGIN
        OLD_STATUS='прете';

        UPDATE KADRY
            SET STATUS='прним'
                WHERE
                    STATUS=:OLD_STATUS
                        AND VREM=:IN_VREM;
    END

END
^

ALTER PROCEDURE PRO_KOLRES (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGN CHAR(20),
OUT_PLAN_R INTEGER,
OUT_FACT_R INTEGER,
OUT_VAKANS INTEGER,
OUT_PRINIM INTEGER,
OUT_NEDOBOR INTEGER)
AS

    BEGIN
    FOR SELECT
        VREM,
    DOLGN,
    PLAN_R,
    FACT_R,
    VAKANS,
    PRINIM,
    NEDOBOR
        FROM STAT
    WHERE VREM = :IN_VREM
    INTO :OUT_VREM,
    :OUT_DOLGN,
    :OUT_PLAN_R,
    :OUT_FACT_R,
    :OUT_VAKANS,
    :OUT_PRINIM,
    :OUT_NEDOBOR
    DO
    BEGIN
```

```
        SELECT COUNT(*)
            FROM KADRY
            WHERE DOLGNOST = :OUT_DOLGN
            AND VREM=:IN_VREM
            AND STATUS='прним'
            INTO :OUT_PRINIM;
            OUT_NEDOBOR=:OUT_VAKANS - :OUT_PRINIM;
            UPDATE STAT
                SET PRINIM=:OUT_PRINIM,
                    NEDOBOR=:OUT_NEDOBOR
                        WHERE VREM = :IN_VREM
                        AND DOLGN=:OUT_DOLGN;
        SUSPEND;
        END
    END
^

ALTER PROCEDURE PRO_RABOT (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
    AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM BETWEEN 1 AND:IN_VREM
    AND STATUS='работ'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
```

```
    :OUT_AVG_BAL,
    : OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRO_PRETE (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
     AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM = :IN_VREM
    AND STATUS='npete'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    : OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^
```

```
ALTER PROCEDURE NACHAT AS

BEGIN
DELETE FROM KADRY;
INSERT INTO KADRY
    SELECT * FROM KADRY_ISH;
DELETE FROM STAT;
INSERT INTO STAT
    SELECT * FROM STAT_ISH;
DELETE FROM RULE;
INSERT INTO RULE
    SELECT * FROM RULE_ISH;
END
^

ALTER PROCEDURE PRO_STAT (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGN CHAR(20),
OUT_PLAN_R INTEGER,
OUT_FACT_R INTEGER,
OUT_VAKANS INTEGER,
OUT_PRINIM INTEGER,
OUT_NEDOBOR INTEGER)
AS

DECLARE VARIABLE VAKANSI INTEGER;
    BEGIN
    FOR SELECT
        VREM,
    DOLGN,
    PLAN_R,
    FACT_R,
    VAKANS,
    PRINIM,
    NEDOBOR
    FROM STAT
    WHERE VREM = :IN_VREM
    INTO :OUT_VREM,
    :OUT_DOLGN,
    :OUT_PLAN_R,
    :OUT_FACT_R,
    :OUT_VAKANS,
    :OUT_PRINIM,
    :OUT_NEDOBOR
    DO
    BEGIN
    SELECT COUNT(*)
```

```
        FROM KADRY
        WHERE DOLGNOST = :OUT_DOLGN
        AND VREM BETWEEN 1 AND :IN_VREM
        INTO :OUT_FACT_R;
        OUT_VAKANS=:OUT_PLAN_R - :OUT_FACT_R;
        UPDATE STAT
            SET FACT_R=:OUT_FACT_R,
                VAKANS=:OUT_VAKANS
                    WHERE VREM = :IN_VREM
                    AND DOLGN=:OUT_DOLGN;
        SUSPEND;
        END
    END
^

ALTER PROCEDURE PRO_PRNJAT (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
    AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM = :IN_VREM
    AND STATUS='принят'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    : OUT_STAG,
```

```
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRO_PINIM (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
    AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM = :IN_VREM
    AND STATUS='прним'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    : OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRO_PROD1 (IN_VREM INTEGER)
AS
```

```
DECLARE VARIABLE OLD_STATUS CHAR(5);

    BEGIN
        OLD_STATUS='принят';
        UPDATE KADRY
        SET STATUS='работ'
            WHERE
                STATUS=:OLD_STATUS
                    AND VREM=:IN_VREM;

END
^

ALTER PROCEDURE PRO_OBJAS (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_DOLGNOST CHAR(20),
OUT_INFORM CHAR(255))
AS

    BEGIN
    FOR SELECT
        K.VREM,
        K.FIO,
        K.STEPEN,
        K.OTKRITIE,
        K.AVG_BAL ,
        K.DOLGNOST,
        R.INFORM
            FROM KADRY K, RULE R
                WHERE VREM=:IN_VREM
                    AND K.NBR_RULE=R.NBR_RULE
INTO
    :OUT_VREM,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    :OUT_DOLGNOST,
    :OUT_INFORM
DO
    SUSPEND;
END
^
```

```
ALTER PROCEDURE PRO_IZMPRAV RETURNS (OUT_NBR_RULE
INTEGER,
OUT_STEPEN CHAR(5),
OUT_OTKRITIE CHAR(5),
OUT_ZNAK_BAL CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_ZNAK_STAZ CHAR(5),
OUT_STAG INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_INFORM CHAR(255))
AS

    BEGIN
        FOR SELECT
            NBR_RULE,
            STEPEN,
            OTKRITIE,
            ZNAK_BAL,
            AVG_BAL,
            ZNAK_STAZ,
            STAG,
            DOLGNOST,
            INFORM
                FROM RULE
INTO
    :OUT_NBR_RULE,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_ZNAK_BAL,
    :OUT_AVG_BAL,
    :OUT_ZNAK_STAZ,
    :OUT_STAG,
    :OUT_DOLGNOST,
    :OUT_INFORM
DO
    SUSPEND;
END
^

ALTER PROCEDURE PRO_IZMRES (IN_VREM INTEGER,
IN_FIO CHAR(15),
IN_STEPEN CHAR(10),
IN_OTKRITIE CHAR(5))
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
```

```
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
        UPDATE KADRY
            SET DOLGNOST='отказать'
                WHERE VREM=:IN_VREM
                    AND STATUS='прним'
                    AND FIO=:IN_FIO
                    AND STEPEN=:IN_STEPEN
                    AND OTKRITIE=:IN_OTKRITIE;
                BEGIN
        FOR SELECT VREM,
        DOLGNOST,
        FIO,
        STEPEN,
        OTKRITIE,
        AVG_BAL,
        STAG,
        STATUS,
        NBR_RULE
        FROM KADRY
        WHERE VREM = :IN_VREM
        AND STATUS='прним'
        INTO :OUT_VREM,
        :OUT_DOLGNOST,
        :OUT_FIO,
        :OUT_STEPEN,
        :OUT_OTKRITIE,
        :OUT_AVG_BAL,
        : OUT_STAG,
        :OUT_STATUS,
        :OUT_NBR_RULE
        DO
        SUSPEND;
    END
END
^

ALTER PROCEDURE PRO_PRINWS (IN_VREM INTEGER)
AS

DECLARE VARIABLE OLD_STATUS CHAR(5);
```

```
    BEGIN
        DELETE FROM KADRY
            WHERE VREM=:IN_VREM
                AND DOLGNOST='отказать';
    BEGIN
        OLD_STATUS='прним';
        UPDATE KADRY
        SET STATUS='принят'
            WHERE
                STATUS=:OLD_STATUS
                    AND VREM=:IN_VREM;
END

END
^

ALTER PROCEDURE PRO_PROD2 AS

BEGIN
DELETE FROM RULE;
INSERT INTO RULE
    SELECT * FROM RULE_ISH;
END
^

ALTER PROCEDURE PRO_KADRYRES (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
    AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
```

```
    FROM KADRY
    WHERE VREM BETWEEN 1 AND :IN_VREM
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    :OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRO_STATRES (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGN CHAR(20),
OUT_PLAN_R INTEGER,
OUT_FACT_R INTEGER,
OUT_VAKANS INTEGER,
OUT_PRINIM INTEGER,
OUT_NEDOBOR INTEGER)
AS

    BEGIN
    FOR SELECT
            VREM,
    DOLGN,
    PLAN_R,
    FACT_R,
    VAKANS,
    PRINIM,
    NEDOBOR
    FROM STAT
    WHERE VREM BETWEEN 1 AND :IN_VREM
    INTO :OUT_VREM,
    :OUT_DOLGN,
    :OUT_PLAN_R,
    :OUT_FACT_R,
    :OUT_VAKANS,
    :OUT_PRINIM,
    :OUT_NEDOBOR
    DO
            SUSPEND;
            END
^
```

```
ALTER PROCEDURE PRO_IZMRES2 (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

        BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
        AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM = :IN_VREM
    AND STATUS = 'прним'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    :OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;

    END
^

ALTER PROCEDURE PRO_RABOT1 (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
```

```
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS
    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
    FIO,
    STEPEN,
    OTKRITIE,
    AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM BETWEEN 1 AND (:IN_VREM - 1)
    AND STATUS='работ'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    : OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRO_UVOLI (IN_VREM INTEGER)
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
    FOR SELECT VREM,
    DOLGNOST,
```

```
    FIO,
    STEPEN,
    OTKRITIE,
    AVG_BAL,
    STAG,
    STATUS,
    NBR_RULE
    FROM KADRY
    WHERE VREM BETWEEN 1 AND (:IN_VREM - 1)
    AND STATUS='уволи'
    INTO :OUT_VREM,
    :OUT_DOLGNOST,
    :OUT_FIO,
    :OUT_STEPEN,
    :OUT_OTKRITIE,
    :OUT_AVG_BAL,
    : OUT_STAG,
    :OUT_STATUS,
    :OUT_NBR_RULE
    DO
    SUSPEND;
    END
^

ALTER PROCEDURE PRO_RABOT2 (IN_VREM INTEGER,
IN_FIO CHAR(15),
IN_STEPEN CHAR(10),
IN_OTKRITIE CHAR(5))
RETURNS (OUT_VREM INTEGER,
OUT_DOLGNOST CHAR(20),
OUT_FIO CHAR(15),
OUT_STEPEN CHAR(10),
OUT_OTKRITIE CHAR(5),
OUT_AVG_BAL NUMERIC(15, 2),
OUT_STAG INTEGER,
OUT_STATUS CHAR(5),
OUT_NBR_RULE INTEGER)
AS

    BEGIN
        UPDATE KADRY
            SET STATUS='уволи'
                WHERE VREM between 1 AND (:IN_VREM -1)

                    AND FIO=:IN_FIO
                    AND STEPEN=:IN_STEPEN
```

```
                    AND OTKRITIE=:IN_OTKRITIE;
                BEGIN
        FOR SELECT VREM,
        DOLGNOST,
        FIO,
        STEPEN,
        OTKRITIE,
        AVG_BAL,
        STAG,
        STATUS,
        NBR_RULE
        FROM KADRY
        WHERE VREM BETWEEN 1 AND (:IN_VREM - 1)
        AND STATUS='работ'
        INTO :OUT_VREM,
        :OUT_DOLGNOST,
        :OUT_FIO,
        :OUT_STEPEN,
        :OUT_OTKRITIE,
        :OUT_AVG_BAL,
        : OUT_STAG,
        :OUT_STATUS,
        :OUT_NBR_RULE
        DO
        SUSPEND;
    END
END
^

ALTER PROCEDURE PRO_STATMIN RETURNS (OUT_MIN INTEGER)
AS

        BEGIN
            FOR SELECT MIN(NEDOBOR)
                FROM STAT
        INTO :OUT_MIN
    DO
        SUSPEND;
END
^
SET TERM ; ^
COMMIT WORK ;
SET AUTODDL ON;

/* Grant permissions for this database */
```

Программа на клиенте

```
program Project1;

uses
    Forms,
    Unit1 in 'Unit1.pas' {Form1},
    Unit2 in 'Unit2.pas' {Form2},
    Unit3 in 'Unit3.pas' {Form3},
    Unit4 in 'Unit4.pas' {Form4},
    Unit5 in 'Unit5.pas' {Form5},
    Unit6 in 'Unit6.pas' {Form6},
    Unit7 in 'Unit7.pas' {Form7},
    Unit8 in 'Unit8.pas' {Form8},
    Unit9 in 'Unit9.pas' {Form9},
    Unit10 in 'Unit10.pas' {Form10},
    Unit11 in 'Unit11.pas' {Form11},
    Unit12 in 'Unit12.pas' {Form12},
    Unit13 in 'Unit13.pas' {Form13},
    Unit14 in 'Unit14.pas' {Form14},
    Unit15 in 'Unit15.pas' {Form15};

{$R *.RES}

begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.CreateForm(TForm2, Form2);
    Application.CreateForm(TForm3, Form3);
    Application.CreateForm(TForm4, Form4);
    Application.CreateForm(TForm5, Form5);
    Application.CreateForm(TForm6, Form6);
    Application.CreateForm(TForm7, Form7);
    Application.CreateForm(TForm8, Form8);
    Application.CreateForm(TForm9, Form9);
    Application.CreateForm(TForm10, Form10);
    Application.CreateForm(TForm11, Form11);
    Application.CreateForm(TForm12, Form12);
    Application.CreateForm(TForm13, Form13);
    Application.CreateForm(TForm14, Form14);
    Application.CreateForm(TForm15, Form15);
    Application.Run;
end.

unit Unit1;
```

```
interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    DBTables, Grids, DBGrids, Db, Menus, StdCtrls;

type
    TForm1 = class(TForm)
    Database1: TDatabase;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    N9: TMenuItem;
    N10: TMenuItem;
    N11: TMenuItem;
    N12: TMenuItem;
    N13: TMenuItem;
    N14: TMenuItem;
    N15: TMenuItem;
    N16: TMenuItem;
    N17: TMenuItem;
    N18: TMenuItem;
    N19: TMenuItem;
    N20: TMenuItem;
    N21: TMenuItem;
    N22: TMenuItem;
    N23: TMenuItem;
    N24: TMenuItem;
    N25: TMenuItem;
    N26: TMenuItem;
    N27: TMenuItem;
    N28: TMenuItem;
    StoredProc1: TStoredProc;
    StoredProc2: TStoredProc;
    N29: TMenuItem;
    StoredProc3: TStoredProc;
    StoredProc4: TStoredProc;
```

```
    StoredProc5: TStoredProc;
    procedure FormCreate(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure N18Click(Sender: TObject);
    procedure N19Click(Sender: TObject);
    procedure N20Click(Sender: TObject);
    procedure N16Click(Sender: TObject);
    procedure N22Click(Sender: TObject);
    procedure N24Click(Sender: TObject);
    procedure N28Click(Sender: TObject);
    procedure N6Click(Sender: TObject);
    procedure N8Click(Sender: TObject);
    procedure N9Click(Sender: TObject);
    procedure N10Click(Sender: TObject);
    procedure N11Click(Sender: TObject);
    procedure N13Click(Sender: TObject);
    procedure N15Click(Sender: TObject);
    procedure N26Click(Sender: TObject);
    procedure N29Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;
    ib1: integer;//изменение правил и результата
    ie1: integer;//недобор отрицателен

implementation
Uses unit3, Unit2, Unit4, Unit5,Unit6, Unit7, Unit8, Unit9, Unit10, Unit11,
Unit12, Unit13,Unit14, Unit15;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
{Начальный доступ}
begin
    N1.Enabled:=True;//ГЛАВНАЯ
    N2.Enabled:=True;//ОБРАБОТАТЬ
    N3.Enabled:=False;// ПОКАЗАТЬ
    N4.Enabled:=True;//Начать
    N6.Enabled:=False;//Запуск правил
    N8.Enabled:=False;//Количественный результат
    N9.Enabled:=False;//Объяснения
    N10.Enabled:=False;//Изменение правил
    N11.Enabled:=False;//Изменение результата
```

```
    N13.Enabled:=False;//Принять всех
    N15.Enabled:=False;//Продолжить
    N16.Enabled:=False;//Штатное расписание
    N18.Enabled:=False;//Кадры
    N19.Enabled:=False;//Работающие
    N20.Enabled:=False;//Претенденты
    N22.Enabled:=False;//Принимаемые
    N24.Enabled:=False;//Принятые
    N26.Enabled:=False;//Уволившиеся
    N27.Enabled:=False;//Открыть
    N28.Enabled:=True;//Завершение
    N29.Enabled:=True;//Выход
ib1:=0;
ie1:=0;
    end;
procedure TForm1.N4Click(Sender: TObject);
{Начать}
begin
    StoredProc1.UnPrepare;
    StoredProc1.Prepare;
    StoredProc1.ExecProc;
    N2.Enabled:=False;//ОБРАБОТАТЬ
    N3.Enabled:=True;//ПОКАЗАТЬ
    N4.Enabled:=False;//Начать
    N16.Enabled:=True;//Штатное расписание
end;

procedure TForm1.N18Click(Sender: TObject);
{Кадры}
begin
Form3.ShowModal;
N19.Enabled:=True;//Работающие
end;

procedure TForm1.N19Click(Sender: TObject);
{Работающие}
begin
Form4.ShowModal;
N20.Enabled:=True;//Претенденты
end;

procedure TForm1.N20Click(Sender: TObject);
{Претенденты}
begin
Form5.ShowModal;
end;
```

```
procedure TForm1.N16Click(Sender: TObject);
{Штатное расписание}
begin
Form2.ShowModal;
N16.Enabled:=False;//Штатное расписание
N18.Enabled:=True;//Кадры
end;

procedure TForm1.N22Click(Sender: TObject);
{Принимаемые}
begin
Form6.ShowModal;
N2.Enabled:=True;//ОБРАБОТАТЬ
N3.Enabled:=False;//ПОКАЗАТЬ
N22.Enabled:=False;//Принимаемые
N8.Enabled:=True//Количественный результат
end;

procedure TForm1.N24Click(Sender: TObject);
{Принятые}
begin
Form11.ShowModal;
N2.Enabled:=True;//ОБРАБОТАТЬ
N3.Enabled:=False;//ПОКАЗАТЬ
N24.Enabled:=False;//Принятые
N15.Enabled:=True;//Продолжить
end;

procedure TForm1.N28Click(Sender: TObject);
{Завершить}
begin
Application.Terminate;
end;

procedure TForm1.N6Click(Sender: TObject);
{Завершить}
begin
StoredProc2.UnPrepare;
StoredProc2.ParamByName('IN_VREM').Value:=StrToInt(Edit1.Text);
StoredProc2.Prepare;
StoredProc2.ExecProc;
N2.Enabled:=False;//ОБРАБОТАТЬ
N3.Enabled:=True;//ПОКАЗАТЬ
N22.Enabled:=True;//Принимаемые
N6.Enabled:=False;//Запуск правил
end;
```

```
procedure TForm1.N8Click(Sender: TObject);
{Количественный результат}
begin
Form7.ShowModal;
if ib1=0 then
begin
if MessageDlg('Объяснения нужны?', mtInformation, [mbYes, mbNo],0) =mrYes
then
begin
N9.Enabled:=True;//Объяснения
N8.Enabled:=False//Количественный результат
end
else
begin
if ie1>=0 then
begin
N8.Enabled:=False;//Количественный результат
N13.Enabled:=True;//Принять всех
end;
if ie1<0 then
begin
ShowMessage('Число принимаемых больше количества вакансий. Измени-
те правила или результаты');
N8.Enabled:=False;//Количественный результат
N10.Enabled:=True;//Изменение правил
N11.Enabled:=True;//Изменение результата
end;
end;
end;
if ib1=1 then
begin
if ie1>=0 then
begin
N8.Enabled:=False;//Количественный результат
N13.Enabled:=True;//Принять всех
end;
if ie1<0 then
begin
ShowMessage('Число принимаемых больше количества вакансий. Измени-
те правила или результаты');
N8.Enabled:=False;//Количественный результат
N10.Enabled:=True;//Изменение правил
N11.Enabled:=True;//Изменение результата
end;
end;
end;
```

```
procedure TForm1.N9Click(Sender: TObject);
{Объяснения}
begin
Form8.ShowModal;
if ie1>=0 then
begin
N9.Enabled:=False;//Объяснения
N13.Enabled:=True;//Принять всех
end;
if ie1<0 then
begin
N9.Enabled:=False;//Объяснения
N10.Enabled:=True;//Изменение правил
N11.Enabled:=True;//Изменение результата
end;
end;

procedure TForm1.N10Click(Sender: TObject);
{Изменение правил}
begin
Form6.ShowModal;
Form9.ShowModal;
if ib1=1 then
begin
N10.Enabled:=False;//Изменение правил
N6.Enabled:=True;//Запуск правил
N11.Enabled:=False;//Изменение результата
end;
end;

procedure TForm1.N11Click(Sender: TObject);
{Изменение результата}
begin
Form10.ShowModal;
if ib1=1 then
begin
N8.Enabled:=True;//Количественный результат
N11.Enabled:=False;//Изменение результата
N10.Enabled:=False;//Изменение правил
end;
end;

procedure TForm1.N13Click(Sender: TObject);
begin
{Принять всех}
StoredProc3.UnPrepare;
StoredProc3.ParamByName('IN_VREM').Value:=StrToInt(Edit1.Text);
```

```
StoredProc3.Prepare;
StoredProc3.ExecProc;
N2.Enabled:=False;//ОБРАБОТАТЬ
N3.Enabled:=True;//ПОКАЗАТЬ
N24.Enabled:=True;//Принятые
N13.Enabled:=False;//Принять всех
end;

procedure TForm1.N15Click(Sender: TObject);
begin
{Продолжение1}
StoredProc4.UnPrepare;
StoredProc4.ParamByName('IN_VREM').Value:=StrToInt(Edit1.Text);
StoredProc4.Prepare;
StoredProc4.ExecProc;
if StrToInt(Form1.Edit1.Text)<StrToInt(Form1.Edit2.Text) then
begin
Form1.Edit1.Text:=IntToStr(StrToInt(Form1.Edit1.Text)+1);
StoredProc5.UnPrepare;
StoredProc5.Prepare;
StoredProc5.ExecProc;
N2.Enabled:=False;//ОБРАБОТАТЬ
N3.Enabled:=True;//ПОКАЗАТЬ
N15.Enabled:=False;//Продолжить
N26.Enabled:=True;//Уволившиеся
Form9.Edit1.Text:='3,5';
Form9.Edit2.Text:=IntToStr(2);
ib1:=0;
ie1:=0;
end
else
begin
ShowMessage('Работа пользователя закончена. Посмотрите результаты');
Form12.ShowModal;
Form13.ShowModal;
ShowMessage('Сеанс окончен');
Application.Terminate;
end;
end;

procedure TForm1.N26Click(Sender: TObject);
{Уволившиеся}
begin
Form15.ShowModal;
Form14.ShowModal;
N26.Enabled:=False;//Уволившиеся
N16.Enabled:=True;//Штатное расписание
end;
```

```
procedure TForm1.N29Click(Sender: TObject);
{Выход}
begin
Close;
end;

end.

unit Unit2;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm2 = class(TForm)
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Button1: TButton;
    Label1: TLabel;
    Query1OUT_VREM: TIntegerField;
    Query1OUT_DOLGN: TStringField;
    Query1OUT_PLAN_R: TIntegerField;
    Query1OUT_FACT_R: TIntegerField;
    Query1OUT_VAKANS: TIntegerField;
    Query1OUT_PRINIM: TIntegerField;
    Query1OUT_NEDOBOR: TIntegerField;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form2: TForm2;

implementation

uses Unit1;

{$R *.DFM}
```

```
procedure TForm2.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm2.Button1Click(Sender: TObject);
begin
Form2.Close;
end;

end.

unit Unit3;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm3 = class(TForm)
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Button1: TButton;
    Label1: TLabel;
    Query1OUT_VREM: TIntegerField;
    Query1OUT_DOLGNOST: TStringField;
    Query1OUT_FIO: TStringField;
    Query1OUT_STEPEN: TStringField;
    Query1OUT_OTKRITIE: TStringField;
    Query1OUT_AVG_BAL: TFloatField;
    Query1OUT_STAG: TIntegerField;
    Query1OUT_STATUS: TStringField;
    Query1OUT_NBR_RULE: TIntegerField;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
```

```
end;

var
    Form3: TForm3;

implementation
Uses unit1;

{$R *.DFM}

procedure TForm3.FormActivate(Sender: TObject);
begin
with Query1 do
begin
    Close;
    ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
    Open;
end;
end;

procedure TForm3.Button1Click(Sender: TObject);
begin
Form3.Close;
end;
end.

unit Unit4;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Grids, DBGrids, Db, DBTables, StdCtrls;

    type
    TForm4 = class(TForm)
    Query1: TQuery;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Button1: TButton;
    Label11: TLabel;
    Query1OUT_VREM: TIntegerField;
    Query1OUT_DOLGNOST: TStringField;
    Query1OUT_FIO: TStringField;
    Query1OUT_STEPEN: TStringField;
    Query1OUT_OTKRITIE: TStringField;
    Query1OUT_AVG_BAL: TFloatField;
```

```
    Query1OUT_STAG: TIntegerField;
    Query1OUT_STATUS: TStringField;
    Query1OUT_NBR_RULE: TIntegerField;
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form4: TForm4;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm4.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm4.Button1Click(Sender: TObject);
begin
Form4.Close;
end;
end.

unit Unit5;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Grids, DBGrids, Db, DBTables, StdCtrls;
type
    TForm5 = class(TForm)
        Query1: TQuery;
```

```
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Label1: TLabel;
        Label2: TLabel;
        Button2: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form5: TForm5;

implementation

uses Unit1;

{$R *.DFM}

    procedure TForm5.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm5.Button1Click(Sender: TObject);
begin
Form5.Close;
end;
```

```
procedure TForm5.Button2Click(Sender: TObject);
begin
{Конец просмотра}
Form1.N2.Enabled:=True;//ОБРАБОТАТЬ
Form1.N3.Enabled:=False;//ПОКАЗАТЬ
Form1.N6.Enabled:=True;//Запуск правил
Form1.N18.Enabled:=False;//Кадры
Form1.N19.Enabled:=False;//Работающие
Form1.N20.Enabled:=False;//Претенденты
Form5.Close;
end;

end.

unit Unit6;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm6 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);

private
    { Private declarations }
public
    { Public declarations }
end;
```

```
var
    Form6: TForm6;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm6.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm6.Button1Click(Sender: TObject);
begin
Form6.Close;
end;

end.

unit Unit7;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm7 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGN: TStringField;
        Query1OUT_PLAN_R: TIntegerField;
        Query1OUT_FACT_R: TIntegerField;
        Query1OUT_VAKANS: TIntegerField;
        Query1OUT_PRINIM: TIntegerField;
```

```
        Query1OUT_NEDOBOR: TIntegerField;
        Query2: TQuery;
        procedure FormActivate(Sender: TObject);
        procedure DBGrid1DrawColumnCell(Sender: TObject; const Rect:
TRect;
            DataCol: Integer; Column: TColumn; State: TGridDrawState);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form7: TForm7;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm7.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
with Query2 do
begin
Close;
Open;
end;
ie1:=Query2.FieldByName('out_min').Value;
end;

procedure TForm7.DBGrid1DrawColumnCell(Sender: TObject; const Rect:
TRect;
    DataCol: Integer; Column: TColumn; State: TGridDrawState);
    {Цвет}
begin
with DBGrid1.Canvas do
begin
If (Query1.FieldByName('Out_nedobor').AsInteger < 0)
```

```
and not (gdSelected in State) then
begin
    Brush.Color:=clRed;
    Font.Color:=clWhite;
    FillRect(Rect);
    TextOut(Rect.Left, Rect.Top, Column.Field.Value);
end//if
Else
    DBGrid1.DefaultDrawColumnCell(Rect, DataCol, Column, State);
end;//with
end;

procedure TForm7.Button1Click(Sender: TObject);
begin
Form7.Close;
end;

end.

unit Unit8;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls, DBCtrls;

type
    TForm8 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        DBMemo1: TDBMemo;
        Label1: TLabel;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_INFORM: TStringField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
```

```
    public
        { Public declarations }
    end;

var
    Form8: TForm8;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm8.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
DBMemo1.DataSource:=DataSource1;
DBMemo1.DataField:='OUT_INFORM';
Open;
end;
end;

procedure TForm8.Button1Click(Sender: TObject);
begin
Form8.Close;
end;

end.

unit Unit9;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;
type
    TForm9 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Edit1: TEdit;
        Label1: TLabel;
```

```
        Label3: TLabel;
        Edit2: TEdit;
        Label2: TLabel;
        Button1: TButton;
        Button2: TButton;
        Label4: TLabel;
        Query1OUT_NBR_RULE: TIntegerField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_ZNAK_BAL: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_ZNAK_STAZ: TStringField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_INFORM: TStringField;
        procedure FormActivate(Sender: TObject);
        procedure Button2Click(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form9: TForm9;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm9.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('select * from pro_izmprav');
Open;
end;
ib1:=0;
end;

procedure TForm9.Button2Click(Sender: TObject);
```

```
begin
Form9.Close;
end;

procedure TForm9.Button1Click(Sender: TObject);
begin
if MessageDlg('Вы уверены в изменении?',mtInformation, [mbYes, mbNo],0)
= mrYes then
begin
if StrToInt(Form9.Edit2.Text) <> 2 then
begin
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('update rule');
SQL.Add('set stag=:stag');
SQL.Add('where stag=2');
ParamByName('stag').Value:=StrToInt(Form9.Edit2.Text);
ExecSQL;
end;//with
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('update Kadry');
SQL.Add('set status=:status');
SQL.Add('where status=:old_status');
SQL.Add('and vrem=:vrem');
ParamByName('status').Value:='прете';
ParamByName('old_status').Value:='прним";
ParamByName('vrem').Value:=StrToInt(Form1.Edit1.Text);
ExecSQL;
ib1:=1;
end;//with
end;//if
if Form9.Edit1.Text <> '3,5' then
begin
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('update rule');
SQL.Add('set avg_bal=:avg_bal');
SQL.Add('where avg_bal=3.5');
ParamByName('avg_bal').AsFloat:=StrToFloat(Form9.Edit1.Text);
```

```
ExecSQL;
end;//with
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('update Kadry');
SQL.Add('set status=:status');
SQL.Add('where status=:old_status');
SQL.Add('and vrem=:vrem');
ParamByName('status').Value:='прете';
ParamByName('old_status').Value:='прним';
ParamByName('vrem').Value:=StrToInt(Form1.Edit1.Text);
ExecSQL;
end;//with
ib1:=1;
end; //if
end;
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('select * from pro_izmprav');
Open;
end;

end;

end.

unit Unit10;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls, Mask, DBCtrls;

type
    TForm10 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
```

```
        Label4: TLabel;
        Button1: TButton;
        Label5: TLabel;
        Label6: TLabel;
        Label7: TLabel;
        Label8: TLabel;
        Label9: TLabel;
        Button2: TButton;
        Label10: TLabel;
        Edit1: TEdit;
        Edit2: TEdit;
        Edit3: TEdit;
        Edit5: TEdit;
        Edit6: TEdit;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form10: TForm10;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm10.FormActivate(Sender: TObject);
{Изменение результата — вывод таблицы}
begin
Edit1.Visible:=False;
Edit2.Visible:=False;
```

```
Edit3.Visible:=False;
Edit5.Visible:=False;
Edit6.Visible:=False;
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('select * from pro_izmres2(:param1)');
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
ib1:=0;
end;

procedure TForm10.Button1Click(Sender: TObject);
{Изменение результата — внесение изменений}
begin
Edit1.Visible:=True;
Edit2.Visible:=True;
Edit3.Visible:=True;
Edit5.Visible:=True;
Edit6.Visible:=True;
Edit1.Text:=Query1.FieldByName('OUT_FIO').Value;
Edit2.Text:=Query1.FieldByName('OUT_STEPEN').Value;
Edit3.Text:=Query1.FieldByName('OUT_OTKRITIE').Value;
Edit5.Text:=Query1.FieldByName('OUT_AVG_BAL').Value;
Edit6.Text:=Query1.FieldByName('OUT_STAG').Value;
if MessageDlg('Вы уверены в изменении?',mtInformation, [mbYes, mbNo],0)
= mrYes then
begin
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('select * from pro_izmres(:param1, :param2, :param3, :param4)');
Query1.ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Query1.ParamByName('param2').Value:=Edit1.Text;
Query1.ParamByName('param3').Value:=Edit2.Text;
Query1.ParamByName('param4').Value:=Edit3.Text;
Query1.Open;
Query1.Locate('OUT_FIO', Edit1.Text, [loPartialKey, loCaseInsensitive]);
ib1:=1;
end;
end;

procedure TForm10.Button2Click(Sender: TObject);
{ДАЛЬШЕ}
begin
```

```
Form10.Close;
end;

end.

unit Unit11;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm11 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form11: TForm11;

implementation

uses Unit1;

{$R *.DFM}
```

```
procedure TForm11.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm11.Button1Click(Sender: TObject);
begin
Form11.Close;
end;

end.

unit Unit12;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm12 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGN: TStringField;
        Query1OUT_PLAN_R: TIntegerField;
        Query1OUT_FACT_R: TIntegerField;
        Query1OUT_VAKANS: TIntegerField;
        Query1OUT_PRINIM: TIntegerField;
        Query1OUT_NEDOBOR: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
```

```
var
    Form12: TForm12;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm12.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm12.Button1Click(Sender: TObject);
begin
Form12.Close;
end;

end.

unit Unit13;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm13 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
```

```
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form13: TForm13;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm13.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm13.Button1Click(Sender: TObject);
begin
Form13.Close;
end;

end.

unit Unit14;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;
type
    TForm14 = class(TForm)
```

```
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Button1: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form14: TForm14;

implementation

uses Unit1;

{$R *.DFM}

procedure TForm14.FormActivate(Sender: TObject);
begin
with Query1 do
begin
Close;
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm14.Button1Click(Sender: TObject);
begin
Form14.Close;
end;

end.
```

```
unit Unit15;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Grids, DBGrids, Db, DBTables, StdCtrls;

type
    TForm15 = class(TForm)
        Query1: TQuery;
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Edit1: TEdit;
        Edit2: TEdit;
        Edit3: TEdit;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Button1: TButton;
        Button2: TButton;
        Query1OUT_VREM: TIntegerField;
        Query1OUT_DOLGNOST: TStringField;
        Query1OUT_FIO: TStringField;
        Query1OUT_STEPEN: TStringField;
        Query1OUT_OTKRITIE: TStringField;
        Query1OUT_AVG_BAL: TFloatField;
        Query1OUT_STAG: TIntegerField;
        Query1OUT_STATUS: TStringField;
        Query1OUT_NBR_RULE: TIntegerField;
        procedure FormActivate(Sender: TObject);
        procedure Button2Click(Sender: TObject);
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form15: TForm15;

implementation

uses Unit1;
```

```
{$R *.DFM}

procedure TForm15.FormActivate(Sender: TObject);
{Уволившиеся — вывод таблицы}
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
with Query1 do
begin
Close;
SQL.Clear;
SQL.Add('select * from pro_rabot1(:param1)');
ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Open;
end;
end;

procedure TForm15.Button2Click(Sender: TObject);
begin
Form15.Close;
end;

procedure TForm15.Button1Click(Sender: TObject);
begin
Edit1.Text:=Query1.FieldByName('OUT_FIO').Value;
Edit2.Text:=Query1.FieldByName('OUT_STEPEN').Value;
Edit3.Text:=Query1.FieldByName('OUT_OTKRITIE').Value;
if MessageDlg('Вы уверены в увольнении?',mtInformation, [mbYes, mbNo],0)
= mrYes then
begin
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('select * from pro_rabot2(:param1, :param2, :param3, :param4)');
Query1.ParamByName('param1').Value:=StrToInt(Form1.Edit1.Text);
Query1.ParamByName('param2').Value:=Edit1.Text;
Query1.ParamByName('param3').Value:=Edit2.Text;
Query1.ParamByName('param4').Value:=Edit3.Text;
Query1.Open;
Query1.Locate("OUT_FIO", Edit1.Text, [loPartialKey, loCaseInsensitive]);
end;
end;

end.
```

Список литературы

1. *Хомоненко А.Д.* и др. Базы данных. — СПб.: Корона принт, 2000. — 416 с.
2. *Конноли Т.* и др. Базы данных: проектирование, реализация, сопровождение. — М.: Вильямс, 2000. — 1120 с.
3. *Саймон А.Р.* Стратегические технологии баз данных: менеджмент на 2000 год. — М.: Финансы и статистика, 1999. — 479 с.
4. *Ульман Д.Д., Уидром Д.* Введение в системы баз данных. — М.: Лори, 2000. — 376 с.
5. *Хансен Г., Хансен Д.* Базы данных: разработка и управление. — М.: Бином, 2000. — 704 с.
6. *Харрингтон Д.Л.* Проектирование баз данных. Просто и доступно. — М.: Лори, 2000. — 224 с.
7. *Григорьев Ю.А., Ревунков Г.И.* Банки данных. — М.: МГТУ, 2002. — 320 с.
8. *Диго С.М.* Проектирование и использование баз данных. — М.: Финансы и статистика, 1995. — 208 с.
9. *Змитрович А.И.* Базы данных. — Минск: Изд-во «Университетское», 1991. — 271 с.
10. *Цикритзис Д., Лоховски Ф.* Модели данных. — М.: 1985. — 344 с.
11. *Дейт К.Д.* Введение в системы баз данных. — М.: Диалектика, 1998. — 820 с.
12. *Мейер Д.* Теория реляционных баз данных. — М.: Мир, 1987. — 608 с.
13. *Грэй П.* Логика, алгебра и базы данных. — М.: Машиностроение, 1989. — 359 с.
14. *Калиниченко Л.А., Рывкин В.М.* Машины баз данных и знаний. — М.: Наука, 1990. — 323 с.
15. *Калиниченко Л.А.* Методы и средства интеграции неоднородных баз данных. — М.: Наука, 1983. — 424 с.
16. *Цегелик Г.С.* Системы распределенных баз данных. — Львов: Свит, 1990. — 167 с.
17. Базы данных и интеллектуальная обработка информации/ Корнеев В.В. и др. — М.: Нолидж, 2000. — 320 с.
18. *Гультеев А.К., Машин В.А.* Проектирование и дизайн пользовательского интерфейса. — СПб.: Корона принт, 2000. — 357 с.
19. *Петров Е.В., Чертовской В.Д.* Компьютерная поддержка экономико-управленческих решений. — М.: Изд-во МГУП, 2001. — 118 с.
20. *Чертовской В.Д.* Особенности обучения объектно-ориентированным технологиям// Материалы 7-й междунар. конференции «Современные технологии обучения». Ч. 1. — СПб.: 2001. С. 153 — 155.
21. *Заикин О.А., Советов Б.Я.* Проектирование интегрированных систем обработки информации и управления. — М.: Мир книги, 1994. — 142 с.
22. *Советов Б.Я., Цехановский В.В.* Автоматизированное управление современным производством. — Л.: Машиностроение, 1988. — 167 с.
23. *Советов Б.Я.* Информационные технологии. — М.: Высш. шк., 1994. — 368 с.

24. *Семенов Л.С.* Анализ информационных объектов на основе связи «система взаимодействующих таблиц». //Автоматика и телемеханика. 1996. № 3. С. 175 — 198.
25. *Калянов Г.Н.* CASE-структурный системный анализ. — М.: Лори, 1996. — 242 с.
26. *Грабер М.* Введение в SQL. — М.: Лори, 1996. — 400 с.
27. *Бекаревич Ю.Б., Пушкина Н.В.* Microsoft Access 2000. — СПб.: BHV, 1999. — 480 с.
28. *Дженнингс Р.* Access™95 в подлиннике. — СПб.: BHV, 1997: В 2 т. Т.1 — 800 с.; т. 2 — 512 с.
29. *Нортон П., Андерсен В.* Разработка приложений в Access-97 в подлиннике. — СПб.: BHV, 1999. — 656 с.
30. Delphi 4. Полное руководство/ Р. Баас, М. Фервай, Г. Гюнтер — СПб.: BHV, 1998. — 800 с.
31. *Тексейра С., Пачеко К.* Delphi 4. Руководство разработчика: В 2 т. — М.: Вильямс, 2000.
32. *Дарахвелидзе П., Марков Е.* Delphi 5. — СПб.: BHV, 2000. — 790 с.
33. *Шумаков П.В.* Delphi 5. Полное руководство пользователя. — М.: Нолидж, 2000. — 690 с.
34. *Александровский А.Д., Шубин В.Д.* Delphi для профессионалов — М.: ДМК, 2000. — 240 с.
35. *Гофман В., Хомоненко А.* Работа с базой данных в Delphi. — СПб.: BHV, 2001. — 656 с.
36. *Эбнер М.* Delphi 5. — Киев: BHV, 2000. — 480 с.
37. *Каратыгин С.А.* и др. Программирование на FoxPro для Windows на примерах. — М.: Бином, 1996. — 493 с.
38. *Кирстен В., Ирингер М., Шульга П.* Объектно-ориентированная разработка приложений в среде постреляционной СУБД САСНЕ. — СПб.: BHV, 2000. — 448 с.
39. *Чертовской В.Д.* Базы и банки данных. — М.: Изд-во МГУП, 2001.
40. *Кречетов Н.Е.* и др. Постреляционная технология САСНЕ для реализации объектных приложений. — М.: МИФИ, 2001. — 152 с.
41. *Культин Н.* Delphi 3. Программирование на Object Pascal. — СПб.: BHV, 1998. — 304 с.
42. *Графф Д.Р., Вайнберг П.Н.* SQL. Полное руководство. — Киев: BHV, 2001. — 816 с.
43. *Галатенко В., Гвоздев А.* Типы и структуры данных в Informix Universal Server. // СУБД. 1997. № 3. С. 54 — 76.
44. *Чертовской В.Д.* Управление предприятием. — Минск: Изд-во «Университетское», 1995. — 263 с.
45. *Чертовской В.Д., Шеховцов О.И.* Объектно-ориентированный подход в построении баз данных. — М.: Изд-во МГУП, 2001. — 64 с.
46. *Ланг К., Чоу Д.* Публикации баз данных в Internet. — СПб.: Символ-Плюс, 1998. — 480 с.
47. Использование HTML4. — Киев: Диалектика, 1999. — 779 с.
48. *Хоумер А., Упмен К.* Dynamic HTML. — СПб.: Питер, 1999. — 510 с.
49. *Гончаров В.* Самоучитель HTML. — СПб.: Питер, 2000. — 239 с.

ОГЛАВЛЕНИЕ

Покупайте наши книги:

В офисе издательства «ЮРАЙТ»:
111123, г. Москва, ул. Плеханова, д. 4а,
тел.: (495) 744-00-12, e-mail: sales@urait.ru, www.urait.ru

В логистическом центре «ЮРАЙТ»:
140053, Московская область, г. Котельники, мкр. Ковровый, д. 37,
тел.: (495) 744-00-12, e-mail: sales@urait.ru, www.urait.ru

В интернет-магазине «ЮРАЙТ»: www.urait-book.ru,
e-mail: order@urait-book.ru, тел.: (495) 742-72-12

Для закупок у Единого поставщика в соответствии
с Федеральным законом от 05.04.2013 № 44-ФЗ (ст. 93)
обращаться по тел.: (495) 744-00-12, e-mail: sales@urait.ru, vuz@urait.ru

Новые издания и дополнительные материалы доступны
в электронной библиотечной системе «Юрайт»
biblio-online.ru

Учебное издание

Советов Борис Яковлевич,
Цехановский Владислав Владимирович,
Чертовской Владимир Дмитриевич

БАЗЫ ДАННЫХ

Учебник для прикладного бакалавриата

Формат 60×90 $^{1}/_{16}$.
Гарнитура «Newton». Печать офсетная.
Усл. печ. л. 28,94. Доп. тираж 500 экз. Заказ №

ООО «Издательство Юрайт»
111123, г. Москва, ул. Плеханова, д. 4а.
Тел.: (495) 744-00-12. E-mail: izdat@urait.ru, www.urait.ru