Министерство образования и науки Российской Федерации

Национальный исследовательский
ядерный университет «МИФИ»

С.Л. Шнырев

БАЗЫ ДАННЫХ

Учебное пособие

Рекомендовано УМО “Ядерные физика и технологии” в качестве учебного пособия для студентов высших учебных заведений

Москва 2011

УДК 004.65(075)
ББК 32.973-018.2я7
Ш 77

Шнырев С.Л. Базы данных: Учебное пособие. М.: НИЯУ МИФИ, 2011. – 224 с.

Посвящено изучению принципов организации и разработки баз данных, включая их типологию, методологические основы, модели и методы доступа, а также основы управления базами данных на уровне поддержки функционирования и администрирования.

В пособии изложены теоретические основы проектирования различных моделей баз данных, в первую очередь реляционных, а также основы администрирования баз данных.

Предназначено для студентов, обучающихся по специальностям «Прикладная информатика», «Прикладная математика и информатика», «Математические методы в экономике», «Экономика и управление на предприятии» и др.

Пособие выполнено в рамках Программы создания и развития НИЯУ МИФИ.

Рецензент: к-т техн. наук *А.Н. Анохин*

ISBN 978-5-7262-1483-2

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Теория баз данных – сравнительно молодая область знаний, ее возраст составляет немногим более 30 лет. Однако она является обязательной для изучения студентами технических и информационных специальностей.

Современный мир информационных технологий невозможен без использования баз данных (БД). Практически все системы в той или иной степени связаны с функциями долговременного хранения и обработки информации. Информация становится фактором, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных, они требуют применения наиболее перспективных компьютерных технологий.

Учебное пособие “Базы данных” относится к блоку общепрофессиональных дисциплин. Основное назначение данного пособия – систематическое введение в идеи и методы, используемые в современных реляционных системах управления базами данных (СУБД). В пособии не рассматривается какая-либо одна популярная СУБД; излагаемый материал в равной степени относится к любой современной системе. Как показывает опыт, без знания основ баз данных трудно на серьезном уровне работать с конкретными системами, как бы хорошо они не были документированы.

Учебное пособие состоит из четырех частей.

Первая часть пособия посвящена общим вопросам современного состояния дел в области автоматизированных информационных систем. Рассматриваются наиболее широко распространенные модели баз данных и перспективы дальнейшего их развития.

Во второй части изложены основы теории одной из наиболее распространенных моделей баз данных – реляционной модели. Описывается математический аппарат, лежащий в основе данной модели – реляционная алгебра, показывается, как этот аппарат используется при проектировании реляционных баз данных.

В третьем разделе рассмотрены основные этапы проектирования баз данных – концептуальный, логический и физический. Исследуются задачи, которые должны быть решены на каждом из

этапов. Рассматриваются современные автоматизированные средства проектирования баз данных.

Четвертый раздел посвящен задачам, которые возникают на различных этапах проектирования баз данных и связаны с необходимостью поддержки логической целостности структур данных.

В пятом разделе рассматриваются вопросы обеспечения надежного функционирования баз данных в процессе их эксплуатации.

Для проверки правильного усвоения материала в конце каждого раздела приведен список контрольных вопросов.

Приведенный список литературы позволит получить более подробную информацию по вопросам, затронутым в данном пособии.

В приложении рассмотрен пример создания и разработки методов управления реляционной БД с помощью современной СУБД Delphi 7.0.

1. ВВЕДЕНИЕ В БАЗЫ ДАННЫХ

1.1. Определение и типология банков данных

В основе решения практически любой задачи лежит обработка определенной информации. Система обработки информации, в том числе и автоматизированная с использованием ЭВМ, называется информационной системой (ИС). Банк данных (БНД) представляет собой разновидность ИС, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации, которая организуется в одну либо нескольких баз данных (БД).

Рис. 1.1. Классификация банков данных

БД является совокупностью специальным образом организованных данных, содержащихся на дисковом пространстве ЭВМ, которые отражают состояние объектов части реального мира и их связей между собой. Сама совокупность этих объектов в их взаимосвязанном состоянии называется *предметной областью (ПО)*.

БНД в общем случае состоит из БД, СУБД, словарей данных, администраторов, вычислительных систем и обслуживающего персонала. В зависимости от выбранных критериев можно выделить различные разновидности БНД (рис. 1.1).

1. *Условия предоставления услуг.* В соответствии с этим критерием различают бесплатные и платные БНД. Последние, в свою очередь, делятся на бесприбыльные (самоокупаемые) и коммерческие.

2. *Обрабатываемая информация.* Так называемые OLTP-системы (On-Line Transaction Processing) предназначены для реализации сравнительно простых запросов к хранимым данным. Напротив, в OLAP-системах (On-Line Analytical Processing) предусмотрены возможности проведения сложных аналитических вычислений.

3. *Степень доступности.* В этом контексте происходит деление БНД на общедоступные и БНД с ограниченным кругом пользователей.

4. *Охват.* Под охватом понимается классификация БНД по территориальному, временному, ведомственному и тематическому признакам.

5. *Характер взаимодействия с пользователем.* В пассивных БНД сам пользователь определяет характер своего взаимодействия с БНД. Активные БНД могут при необходимости сами менять свое "поведение".

6. *Форма собственности.* БНД могут быть государственными, частными и личными.

К БНД всегда предъявляются общие требования, которые заключаются в следующем:

- простота и легкость использования, т. е. возможность пользователей легко распознавать данные, доступ к которым должен быть простым;
- многократное использование данных – различные пользователи могут использовать одни и те же данные по-разному;
- гибкость использования данных – одни и те же данные могут быть получены пользователем посредством различных критериев запроса;
- быстрая обработка запросов к данным;
- обеспечение контроля целостности данных;

- возможность восстановления данных после сбоев.

В основе любого БНД лежит информационная база – данные, отражающие состояние данной предметной области. Информационная база состоит из собственно данных и описания этих данных (метаданных). Отделение данных от их описания объясняется тем, что в зависимости от различных условий одни и те же данные могут представляться и использоваться по-разному. Кроме того, в разных прикладных задачах требуются различные наборы данных, совокупность которых обеспечивает полноту информации. Это означает, что сами данные и связи между ними могут быть описаны и представлены различными способами.

В настоящее время существует стандарт ANSI (American National Standards Institute), в соответствии с которым имеется три уровня представления данных – физический (внутренний) уровень, концептуальный уровень и уровень внешних моделей.

Под физическим уровнем понимаются собственно данные, специальным образом организованные и размещенные в файловых либо странично-сегментных структурах на внешних носителях.

Концептуальный уровень является наиболее общим. На этом уровне отражается обобщенная модель предметной области.

Уровень внешних моделей отражает особенности видения данных отдельными приложениями. При этом каждое приложение имеет доступ только к тем данным, которые ему необходимы.

Любой БНД имеет определенные стадии своего существования: проектирование, реализация, эксплуатация, модернизация и развитие, реорганизация. На каждом из этих этапов с БНД работает определенная категория пользователей. Основными категориями пользователей являются следующие.

1. *Конечные пользователи.* Это основная категория пользователей, именно для нее в конечном счете и создается БНД. Эти пользователи могут быть случайными (например, клиенты, просматривающие электронный каталог фирмы) и регулярными (например, сотрудники этой фирмы, в чьи служебные обязанности входит работа с данными, содержащимися в БНД, на уровне выполнения запросов).

2. *Администраторы БНД.* Эта группа пользователей работает с БНД на всех этапах его существования. Администраторы, в частно-

сти, обеспечивают оптимальную организацию БНД в многопользовательском режиме, отвечают за поддержку целостности БНД, и т. д.

3. *Разработчики и администраторы приложений.* Эта категория пользователей также работает с БНД на всех этапах его существования и отвечает за разработку специализированных программ-приложений для обработки данных.

БД является ядром любого БНД. Имеется множество признаков БД, в соответствии с которыми может быть проведена их классификация.

1. По *форме представления информации* БД делятся на визуальные БД, аудиосистемы и средства мультимедиа. Информация, хранимая в БД, может быть представлена в виде разном виде – в виде изображений (рисунки, чертежи и схемы, фотографии, движущиеся изображения, анимация), звука и т. д.

2. По *характеру организации данных* БД могут быть неструктурированными, частично-структурированными и структурированными.

Эта классификация относится к символьным БД. Неструктурированными называются БД, информация в которых представлена в виде так называемых семантических сетей. Частично-структурированные БД содержат информацию в виде текста. В структурированных БД перед заполнением их данными должна быть предварительно описана модель их структуры. В зависимости от типа используемой модели структурированные БД делятся на иерархические, сетевые, реляционные, постреляционные, многомерные и объектно-ориентированные.

3. По *типу хранимой информации* БД делятся на документальные, фактографические и лексикографические.

Документальные БД являются частично-структурированными и ориентированы, главным образом, на хранение текстовых данных в различных форматах. Информационной единицей в документальных БД является документ-текст. Среди этих моделей выделяют библиографические, реферативные и полнотекстовые модели.

Лексикографические модели организованы на принципах организации словарей и содержат в себе определенные языковые кон-

струкции. Основное назначение этих моделей – использование в системах-переводчиках.

Фактографические модели являются структурированными и в зависимости от способа структуризации делятся на теоретико-графовые (иерархическая и сетевая модели), теоретико-множественные (реляционная, постреляционная и многомерная) и объектно-ориентированные.

4. По *характеру организации хранения* данных БД бывают персональными и распределенными.

Персональные БД предназначены для одного конкретного пользователя. Распределенные БД предполагают возможность одновременного обращения к данным со стороны множества пользователей.

СУБД представляет собой комплекс программных средств, предназначенный для создания, обслуживания и использования БД в совместном режиме множеством пользователей. Помимо СУБД, существуют специализированные программы-приложения, которые служат для автоматизированной обработки информации для какой-либо прикладной задачи.

СУБД так же, как и БД, делятся на ряд категорий.

1. По *языкам общения* СУБД делятся на открытые и замкнутые.

В открытых СУБД работа с данными осуществляется с использованием распространенных общепринятых универсальных языков программирования. В замкнутых СУБД используют собственные языки. Смешанные СУБД сочетают в себе частично свойства открытых и замкнутых СУБД

2. В зависимости от *количества звеньев* СУБД бывают одно-, двух- и трехзвенными. Категоризация СУБД по этому принципу соответствует понятию архитектуры БД.

В однозвенной БД единственным звеном является клиент. При двухзвенной архитектуре появляется новое звено – сервер БД, которое функционально предназначено для обеспечения части функций логического управления данными и их визуализации. Роль клиента заключается в обеспечении удобного с его точки зрения способа отображения данных. В трехзвенных СУБД появляется сервер приложений, который фактически является промежуточным звеном между клиентом и сервером БД. Его назначение заключает-

ся в обеспечении управления данными (клиент полностью освобождается от этого) и обеспечении связи клиента с сервером БД.

3. В зависимости от *физического расположения* различают локальные и сетевые СУБД.

Локальная СУБД целиком размещается на компьютере пользователя. Если таких пользователей несколько, то каждый из них должен иметь свою локальную копию СУБД.

Сетевые СУБД делятся на файл-серверные, клиент-серверные и распределенные.

В файл-серверной модели как СУБД, так и БД, как правило, размещаются на одном компьютере, который называется файл-сервером. Пользователи получают доступ к информации со своих персональных компьютеров (клиентские места) посредством развертывания локальной сети. Таким образом, между локальными и файл-серверными вариантами принципиальных различий нет.

Клиент-серверные СУБД являются фактически двухзвенными, поскольку в этом случае часть СУБД размещается на сервере БД. Эта часть СУБД отвечает за получение запроса от клиента, отыскание в данных нужной информации и передачу ее клиенту.

Распределенные СУБД могут размещаться на десятках и сотнях серверов БД.

4. По *выполняемым функциям* СУБД делятся на информационные и операционные.

Информационные СУБД обеспечивают хранение данных и доступ к ним. Операционные СУБД предназначены для более сложной обработки информации, например, для проведения вычислений с привлечением данных, не содержащихся в БД.

5. В зависимости от *сферы использования* различают универсальные и специализированные СУБД.

6. По *мощности* СУБД делятся на настольные и корпоративные.

К настольным СУБД предъявляются невысокие требования с точки зрения технических средств и стоимости. Корпоративные СУБД предназначены для распределенной обработки данных и имеют, в частности, развитые средства автоматизированного администрирования и обеспечения целостности.

Кроме того, СУБД можно классифицировать в зависимости от того, для какой категории пользователей они предназначены. В

этом контексте СУБД могут быть ориентированы на разработчиков и на конечных пользователей.

Обеспечение взаимодействия любой из категорий пользователей с информацией, хранящейся в БД, осуществляется всегда с использованием языковых средств данной СУБД. Существующие разновидности СУБД используют обширные и разнообразные наборы языковых средств, которые можно разделить на различные категории, представленные на рис. 1.2.

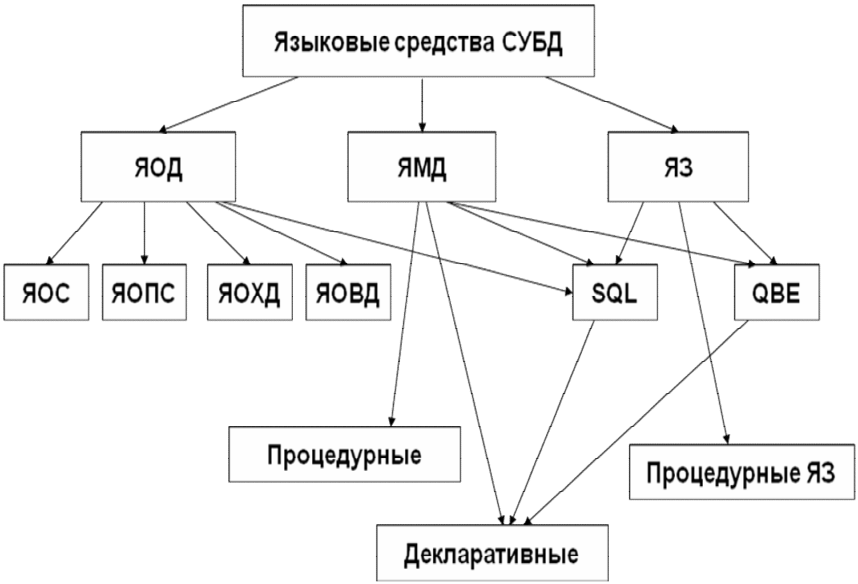


Рис. 1.2. Языковые средства современных СУБД

1. *Языки описания данных* (ЯОД). В эту группу языков входят языки описания схем (ЯОС) и подсхем данных (ЯОПС), языки описания хранимых данных (ЯОХД), языки описания внешних данных (ЯОВД).

2. *Языки манипулирования данными* (ЯМД). Эти языки в свою очередь делятся на процедурные и непроцедурные. При использовании процедурных языков пользователь сам программно разрабатывает способ, который обеспечивает нужный для него результат. Непроцедурные языки являются декларативными.

3. *Языки запросов* (ЯЗ). К языкам запросов относятся языки, непосредственно имеющие только лишь соответствующие возможности, а также комплексные языки запросов, которые имеют еще и возможности обновления данных.

Часто бывает, что для одной и той же цели можно использовать языки различных типов. Такими языками являются, в частности, табличный язык запросов QBE и язык SQL.

Формы представления данных языковыми средствами СУБД перечислены на рис. 1.3.

Рис. 1.3. Формы представления данных языковыми средствами СУБД

1.2. Уровни и типы моделей баз данных

Любая БД отражает информацию об определенной предметной области. В зависимости от уровня абстракции, на котором представляется предметная область, существуют различные уровни моделей данных (рис. 1.4). Под информационной моделью данных подразумевается способ описания информации, содержащейся в предметной области. В дальнейшем будут рассматриваться структурированные модели данных. Для этих моделей существует четы-

ре основных уровня моделей: концептуальный, даталогический или логический, физический и уровень внешних моделей (рис. 1.5).

Рис. 1.4. Модели баз данных

На первом уровне описание предметной области строится так, чтобы оно было как можно более общим, не зависело от особенностей выбираемой впоследствии СУБД, а информация была бы доступна широкой категории пользователей: от заказчиков до системных программистов, которые будут заниматься проектированием БД на основе этой модели. Для этого исходная информация о предметной области анализируется и представляется в некотором формализованном виде. Это формализованное описание предметной области должно отражать ее специфику и использоваться на следующих этапах проектирования структуры БД в контексте особенностей выбранной конкретной СУБД. Такое формализованное описание предметной области называется концептуальной моделью.

Затем строится модель в терминах конкретной СУБД, выбранной для проектирования БД. Этот уровень называется даталогической (логической) моделью. Описание логической структуры БД на языке выбранной СУБД называется ее схемой.

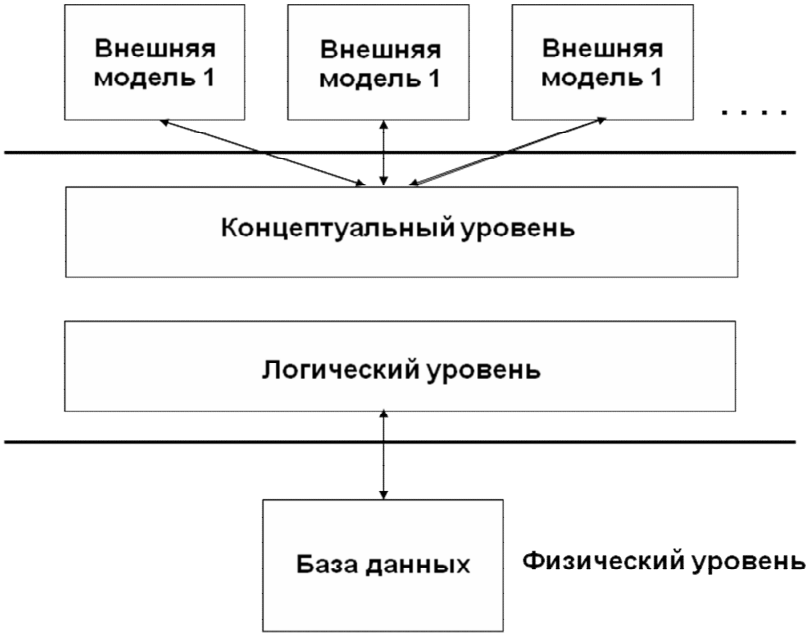


Рис. 1.5. Четырехуровневая модель представления данных

Следующим уровнем является физическая модель данных. В рамках этой модели определяются способы физического размещения данных в среде хранения, разрабатывается так называемая схема хранения данных. Поскольку в разных СУБД имеются различные возможности и особенности физической организации данных, то физическое моделирование проводится только после разработки логической модели.

Ряд современных СУБД обладают возможностями описания структуры БД с точки зрения конкретного пользователя. Такое описание называется внешней моделью. Для каждого типа пользователей внешнее моделирование позволяет разработать подсхему БД исходя из потребностей различных категорий пользователей.

Этот подход является удобным с точки зрения облегчения работы пользователей с БД, поскольку пользователь при этом может, не зная о всей структуре БД, работать только с той ее частью, которая имеет к нему непосредственное отношение. Кроме того, механизм создания подсхем служит дополнительным средством защиты информации, хранимой в БД.

Таким образом, если СУБД поддерживает возможность создания подсхем, то архитектура БД становится трехуровневой: уровень схемы хранения, уровень схемы и уровень подсхем.

Рассмотрим теперь основные типы фактографических моделей данных, представленных на рис. 1.6.

Рис. 1.6. Фактографические модели баз данных

Иерархическая модель БД является одной из первых моделей БД. Это обусловлено прежде всего тем, что именно такая модель наиболее естественным образом отражает множественные связи между объектами реального мира, когда один объект выступает в качестве главного (родительского), с которым связано большое количество подчиненных (дочерних) объектов.

Принцип иерархической модели БД заключается в том, что все связи между данными описываются с помощью построения упорядоченного графа или дерева (рис. 1.7).

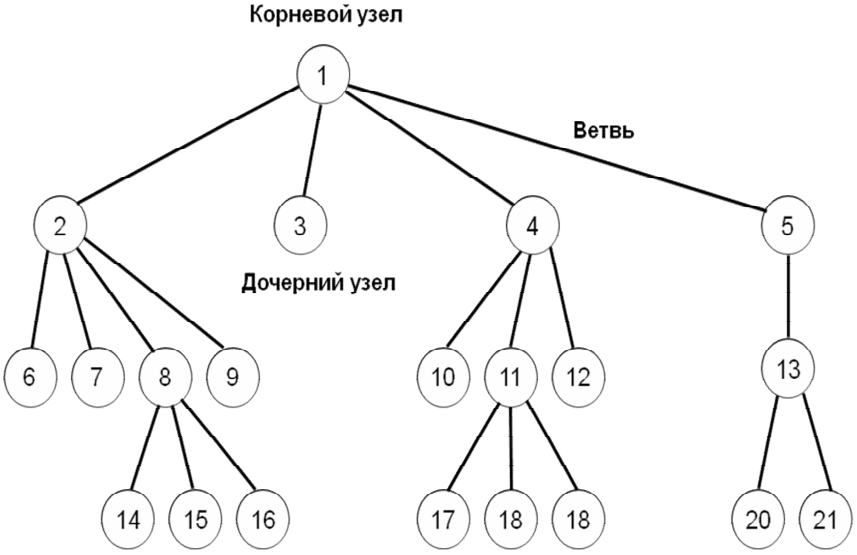


Рис. 1.7. Упорядоченный граф – древовидная структура

Дерево является упорядоченным в соответствии с иерархией наборов элементов, которые называются узлами. Все узлы связаны между собой ветвями. При этом для описания схемы иерархической БД понятие “дерево” используется как определенный тип данных. Этот тип данных является составным и может включать в себя подтипы или поддеревья. БД является совокупностью деревьев, каждое из которых на языке иерархической модели называется физической базой данных. Каждое дерево состоит из единственного корневого (главного, родительского) типа или узла и связанного с ним упорядоченного множества подчиненных (дочерних) типов. Корневой тип – это такой тип, который имеет подчиненные типы и не имеет родительских. Дочерние типы, имеющие один и тот же родительский тип, называются близнецами. Каждый из подчиненных типов для данного корневого типа может являться как простым, так и составным типом “запись”.

Различают три вида деревьев – сбалансированные, несбалансированные и двоичные. В сбалансированном дереве (рис. 1.8а) каждый узел имеет одно и то же количество ветвей. Такая организация данных физически является наиболее простой, однако часто логическая структура данных требует переменного количества ветвей в каждом узле, что соответствует несбалансированному дереву (рис. 1.8б). Двоичные деревья допускают наличие не более двух ветвей для одного узла.

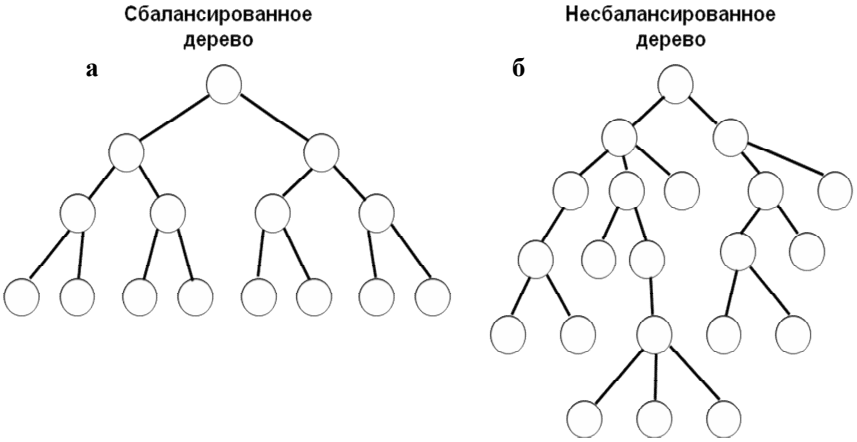


Рис. 1.8. Примеры сбалансированного и несбалансированного графов

Таким образом, иерархическая модель БД может быть интерпретирована как упорядоченная совокупность экземпляров деревьев, каждое из которых содержит экземпляры записей. Собственно содержание БД хранится в полях записей. Под полем записи понимается минимальная, неделимая единица данных.

При построении иерархической модели БД всегда необходимо помнить о поддержке целостностей связей, подразумевая под этим, что:

- всегда имеется по крайней мере один родительский тип, который может иметь произвольное количество подчиненных типов;
- дочерние типы не могут существовать без наличия родительского типа, причем для каждого подчиненного типа в БД имеется единственный корневой тир;

- у корневого типа не обязательно должны иметься подчиненные типы.

Необходимо отметить, что в ряде нотаций может использоваться иная терминология. Так, в нотации Американской Ассоциации по базам данных DBTG (Data Base Task Group) термину “запись” соответствует термин “сегмент”, а записью называется все множество записей, которые относятся к одному экземпляру типа “дерево”.

Основным достоинством иерархической модели БД является относительно высокая скорость обработки информации при обращении к данным. К недостаткам следует отнести ее громоздкость при наличии сложных логических связей между данными.

Пример иерархической модели приведен на рис. 1.9 и рис. 1.10.

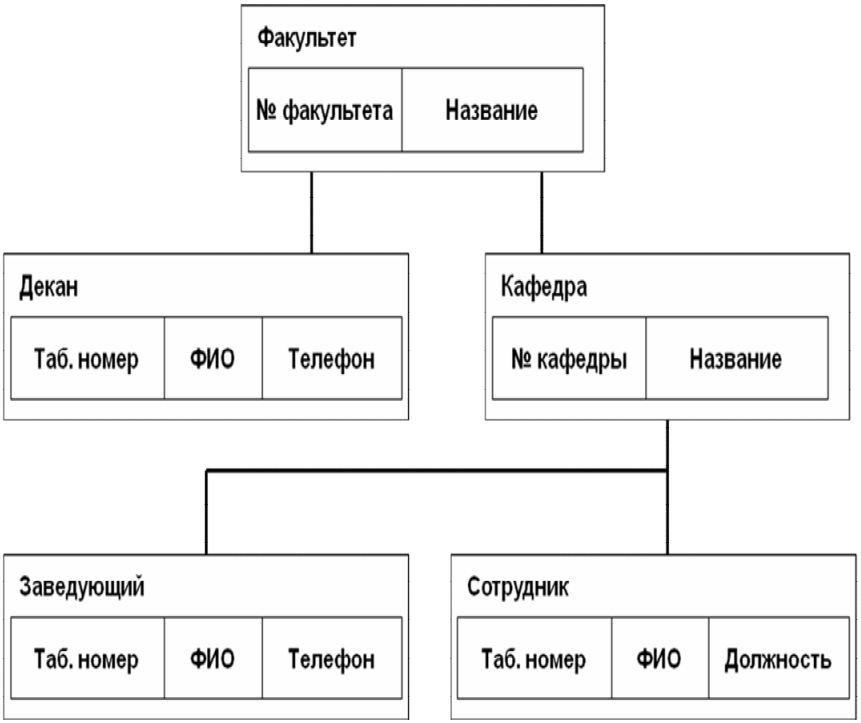


Рис. 1.9. Пример структуры иерархической модели

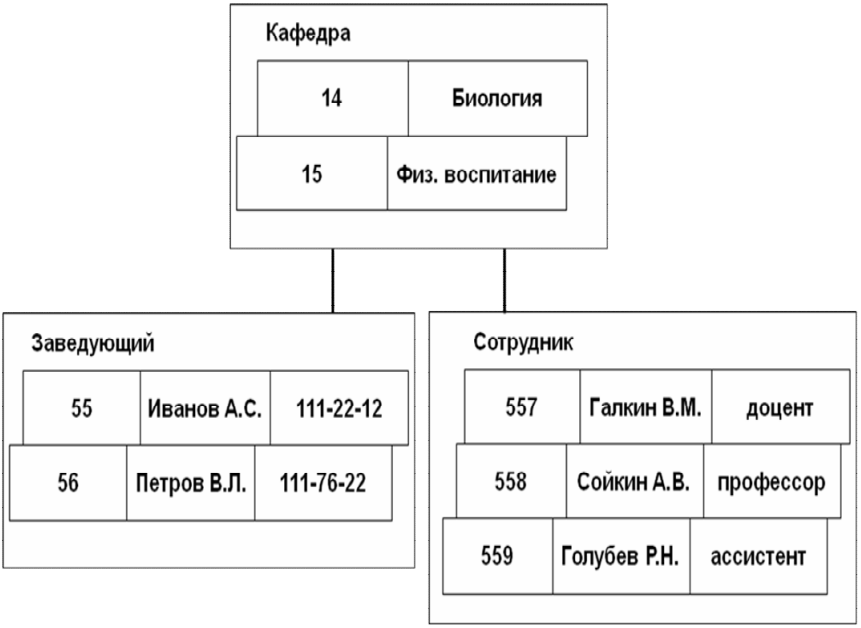


Рис. 1.10. Пример представления данных в иерархической модели

Сетевая модель БД является в некотором смысле обобщением иерархической модели. Основное отличие сетевой модели от иерархической заключается в том, что в сетевой модели подчиненный тип может иметь произвольное количество родительских типов (рис. 1.11).

Основными понятиями сетевой модели являются набор, агрегат, запись и элемент данных. Под элементом данных в данном случае следует подразумевать то же самое, что и в иерархической модели – минимальную единицу данных. Агрегаты данных бывают двух типов: агрегат типа вектор и агрегат типа повторяющаяся группа (рис. 1.12). Агрегат типа вектор соответствует набору элементов данных. Агрегат типа повторяющаяся группа соответствует совокупности векторов данных. Записью называется совокупность агрегатов данных. Каждая запись имеет определенный тип и состоит из совокупности экземпляров записи. Набором называется граф, связывающий два типа записи. Таким образом, набор отражает иерархическую связь между двумя типами записей. Родительский тип

записи в данном наборе называется владельцем набора, а дочерний тип записи – членом того же набора.

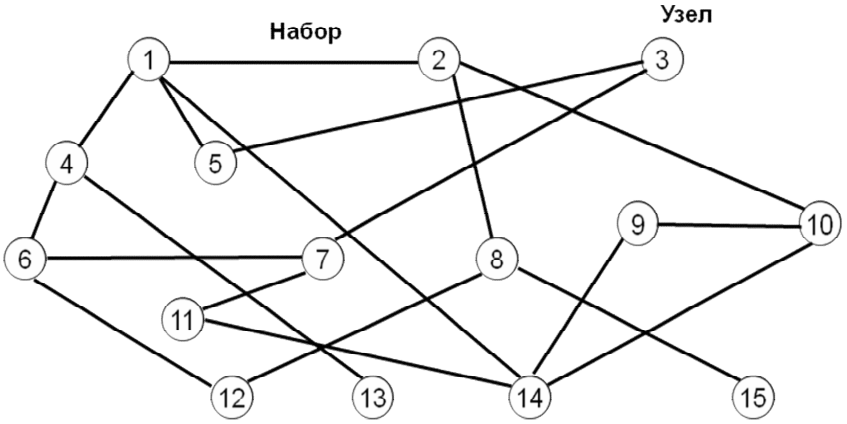


Рис. 1.11. Пример сетевой модели

Рис. 1.12. Агрегаты данных в сетевой модели

Для каких-либо любых двух типов записей может быть задано любое количество связывающих их наборов. При этом между дву-

мя типами записей может быть определено различное количество наборов. Однако один и тот же тип записи не может быть одновременно владельцем и членом набора.

Несомненным достоинством сетевой модели данных является возможность более гибкого отображения множественных связей между объектами. Один из наиболее существенных недостатков заключается в высокой сложности схемы построения БД, что усугубляется ослаблением контроля за целостностью связей ввиду их многочисленности.

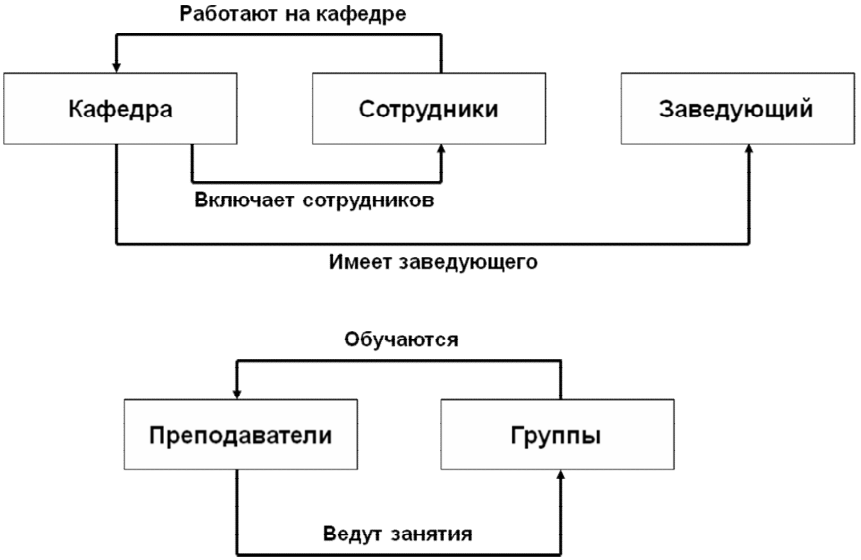


Рис. 1.13. Примеры структур данных в сетевой модели

Примеры структуры сетевой модели приведены на рис. 1.13.

В основе *реляционной модели* данных лежит понятие отношения (рис. 1.14), которое является двумерной таблицей, содержащей множество строк (*кортежей*) и столбцов (*полей* или *атрибутов*). Таблица соответствует определенному объекту предметной области, ее поля описывают свойство данного объекта, а строки – конкретные экземпляры объекта. В каждом отношении всегда должен присутствовать атрибут или набор атрибутов, однозначно определяющий единственный кортеж этого отношения – *первичный ключ*.

Для отражения связи между объектами используется связывание таблиц по определенным правилам с использованием так называемых *внешних ключей*, которые будут подробно рассмотрены в следующих разделах.

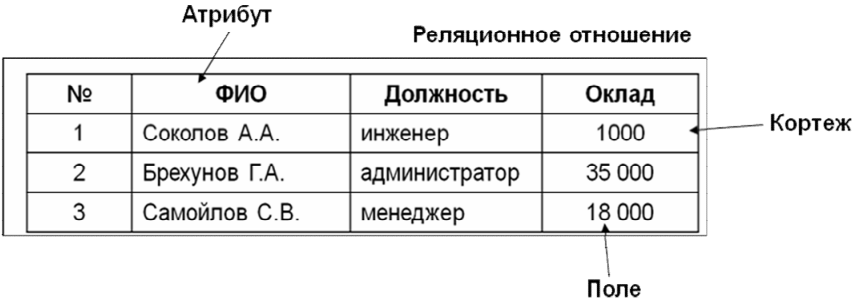


Рис. 1.14. Пример реляционного отношения

Основное достоинство реляционной модели заключается в ее простоте и логической замкнутости, а недостатком является сложность системы описания различных связей между таблицами.

Развитие реляционной модели привело к появлению так называемой *постреляционной модели* данных, основным отличием которой является допустимость многозначных полей (полей, значения которых состоят из множества подзначений). Многозначные поля можно интерпретировать как самостоятельные таблицы, встроенные в исходную таблицу. Кроме того, в постреляционной модели поддерживаются множественные ассоциированные поля, в совокупности образующих ассоциацию: в каждой строке первое значение одного столбца ассоциации соответствует первым значениям всех остальных столбцов ассоциации.

Рис. 1.15 иллюстрирует сравнение представления данных в реляционной и постреляционной моделях.

Основное достоинство постреляционной модели заключается в том, что она позволяет более эффективно хранить данные, а количество таблиц в этой модели заметно меньше по сравнению с реляционной. Недостатком является сложность обеспечения поддержания логической согласованности данных.

Теория *многомерных моделей* данных активно развивается в последнее время. Понятие многомерной модели означает многомер-

ность логического представления структуры информации. Основными понятиями многомерной модели являются измерение и ячейка.

Данные в реляционной модели

Таб_№	ФИО
12	Ежов В.Е.
15	Денисов А.Б.
16	Козлов С.Л.

Таб_№	Предмет	Часы
12	Паскаль	32
12	C++	48
15	Дельфи	32
16	Oracle	32
16	MySQL	64

Данные в постреляционной модели

Таб_№	ФИО	Предмет	Часы
12	Ежов В.Е.	Паскаль	32
		C++	48
15	Денисов А.Б.	Дельфи	32
16	Козлов С.Л.	Oracle	32
		MySQL	64

Рис. 1.15. Сравнение способов представления данных в реляционной и постреляционной моделях

Измерением называется множество данных одного типа, которые образуют грань *n*-мерного куба. На рис. 1.16 в качестве примера приведен трехмерный куб. Ячейкой является поле, значение которого определяется всей совокупностью измерений. Значение ячейки может быть переменной или формулой.

Для работы с многомерными моделями данных используются специальные многомерные СУБД, в основе которых лежат понятия агрегируемости, историчности и прогнозируемости. Под *агрегируемостью* данных подразумеваются различные уровни обобщения информации. *Историчность* данных означает высокий уровень статичности как самих данных, так и связей между ними, а также упорядочение данных во времени в процессе их обработки и пред-

ставления пользователям. Обеспечение *прогнозируемости* задается использованием специальных функций прогнозирования.

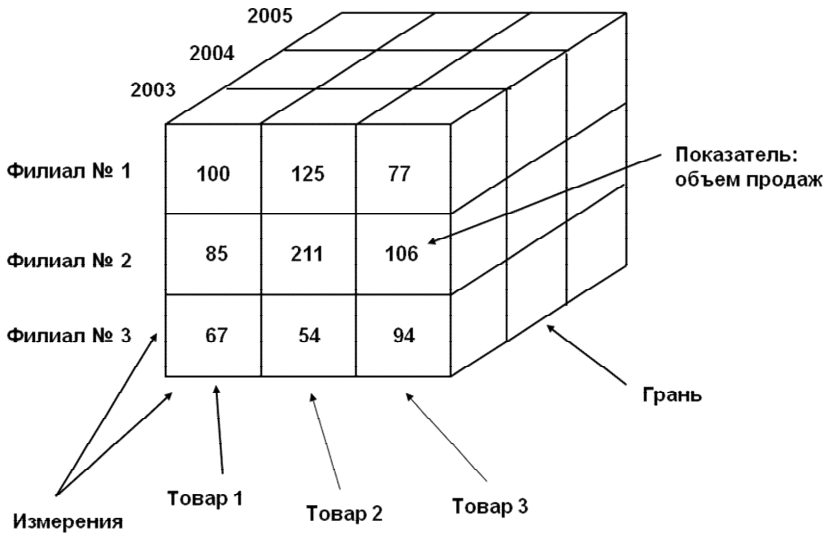


Рис. 1.16. Трехмерный куб в многомерной модели

Многомерные СУБД используют две схемы организации данных – поликубическую и гиперкубическую. В поликубической модели *n*-мерные кубы могут иметь как различные размерности, так и различные измерения-грани. В гиперкубической модели все размерности кубов одинаковы, а измерения различных кубов совпадают.

Срезом называется некоторое подмножество *n*-мерного куба, задаваемое фиксацией заданного количества измерений. Срез имеет размерность, меньшую *n*, и используется, в частности, для представления информации пользователям в виде читаемых двумерных таблиц. Вращение также часто используется для двумерного представления данных и заключается в изменении порядка измерений. Операции агрегации и детализации означают более общее или более детальное представление информации.

Многомерные модели данных особенно удобны для работы с большими БД, поскольку позволяют эффективно обрабатывать

значительные объемы информации, и это является их несомненным достоинством.

Данные в реляционной модели

Товар	Год	Количество
Товар 1	2005	120
Товар 1	2006	125
Товар 1	2007	132
Товар 2	2005	110
Товар 2	2006	115
Товар 2	2007	123

Данные в многомерной модели

Товар	2005	2006	2007
Товар 1	120	125	132
Товар 2	110	115	123

Рис. 1.17. Сравнение способов представления данных в реляционной и многомерной моделях

Рис. 1.17 иллюстрирует сравнение представления данных в реляционной и многомерной моделях.

Основным отличием *объектно-ориентированной модели* от рассмотренных выше является использование объектно-ориентированных методов манипулирования данными – инкапсуляции, наследования и полиформизма.

Инкапсуляция означает возможность разграничения доступа различных программ, приложений, методов и функций (в более широком смысле и доступа различных категорий пользователей) к различным свойствам объектов данных. В контексте термина “инкапсуляция” часто используется понятие видимости – степень доступности отдельных свойств объекта. В современных объектно-ориентированных системах программирования (таких как Delphi или C++ Builder) имеются следующие уровни инкапсуляции (видимости), которые принято называть разделами:

1. Разделы *Public*, *Published* и *Automated* – с незначительными отличительными особенностями свойства объекта, описанные как принадлежащие к данным разделам, полностью доступны.

2. Раздел *Private* – этот раздел накладывает наиболее жесткие ограничения на видимость свойств объекта. Как правило, такие свойства оказываются доступными только владельцу данного объекта (программному модулю, в котором этот объект создан).

3. Раздел *Protected* – в отличие от раздела *Private* свойства объекта становятся доступными наследникам владельца объекта.

В отличие от инкапсуляции *наследование* предполагает полную передачу всех свойств родительского объекта дочерним объектам. При необходимости наследование свойств одного объекта можно распространить и на объекты, не являющиеся по отношению к нему дочерними.

Полиморфизм означает возможность одного и того же приложения манипулировать с данными разных типов – приложения (методы, процедуры и функции), обрабатывающие объекты различных типов, могут иметь одно и то же имя.

Основным достоинством объектно-ориентированых моделей является возможность моделировать разнообразные сложные взаимосвязи между объектами.

Контрольные вопросы

1. Что называется банком данных и чем он отличается от базы данных?
2. Что такое система управления базами данных?
3. Какие существуют уровни представления данных в соответствии со стандартом ANSI?
4. Перечислите основные категории пользователей баз данных.
5. Какими бывают базы данных с точки зрения формы представления информации?
6. Какие структурированные, частично-структурированные и неструктурированные модели баз данных вы можете назвать?
7. Какими бывают базы данных с точки зрения типа хранимой информации?
8. Чем отличаются открытые СУБД от замкнутых?

9. В чем заключается различие между однозвенными, двух- и трехзвенными СУБД?

10. Какие СУБД называются локальными, а какие – сетевыми?

11. В чем заключается различие между файл-серверными, клиент-серверными и распределенными СУБД?

12. Какие языковые средства, используемые в современных СУБД, вы можете назвать? В чем заключается предназначение этих языковых средств?

13. Что называется предметной областью базы данных?

14. В чем заключаются основные принципы построения иерархической и сетевой моделей? В чем их сходство и чем они принципиально различаются?

15. В чем заключаются основные преимущества реляционной модели баз данных по сравнению с более ранними моделями?

16. Опишите основные отличительные особенности постреляционной и многомерной моделей баз данных.

17. Дайте определения понятиям агрегируемости, историчности и прогнозируемости.

18. Что такое сбалансированное дерево? Чем оно отличается от несбалансированного и двоичного деревьев?

19. Что называется агрегатом данных?

20. Что такое родительский и дочерний типы данных?

21. Дайте определения понятиям инкапсуляции, наследования и полиморфизма.

22. Какие схемы организации данных используются в многомерных моделях?

23. Что из себя представляет ассоциированное поле, используемое в постреляционной модели?

24. Чем отличаются настольные СУБД от корпоративных?

2. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

2.1. Построение реляционной схемы базы данных

Как уже говорилось, основным понятием реляционной модели является *отношение* (*relation* – отсюда и название модели), представляющее собой двумерную таблицу, содержащую некоторые данные. Реляционной моделью данных является совокупность отношений, изменяющихся во времени. Эта совокупность отношений хранит информацию об объектах некоторой предметной области и связях между ними.

Для математического определения отношения рассмотрим n множеств D_1, D_2, …, D_n, которые в реляционной модели называются *доменами*. Полным декартовым произведением будет называться набор всевозможных сочетаний из n элементов каждое, причем каждый из элементов берется из своего домена. Отношением будет называться подмножество полного декартова произведения – множество упорядоченных кортежей $\{d_1 \in D_1,\ d_2 \in D_2,\ \ldots,\ d_n \in D_n\}$, где элемент d_i называется атрибутом отношения.

Важно обратить внимание на то, что не всякой двумерной таблице можно поставить в соответствие отношение. Для этого необходимо обеспечить выполнение следующих условий.

1. В таблице не может быть двух одинаковых строк, что означает, что в отношении не может быть двух одинаковых значений первичного ключа.

2. Столбцы таблицы должны соответствовать атрибутам отношения, то есть имена столбцов должны быть различны, поскольку каждый атрибут в отношении имеет уникальное имя.

3. Порядок строк в таблице может быть произвольным.

4. Все строки в таблице должны иметь одинаковую структуру.

Схемой отношения называется список атрибутов этого отношения. Количество атрибутов, входящих в кортежи отношения, называется его *степенью*. Первичным ключом отношения называется набор атрибутов, однозначно идентифицирующий каждый его кортеж. Если такой ключ состоит из одного атрибута, то он называется простым, в противном случае – составным.

Любое отношение имеет первичный ключ, поскольку по определению в отношении не может быть двух одинаковых кортежей. В самом предельном случае таким ключом может являться полный набор атрибутов отношения. В случае, если в отношении имеется несколько неодинаковых наборов атрибутов, однозначно определяющих его кортежи, то каждый из таких наборов называется *возможным ключом* отношения. Если в отношении имеется несколько возможных ключей, то в качестве первичного ключа выбирается один из них. При этом если первичный ключ состоит из минимального необходимого количества атрибутов, то такой первичный ключ называется неизбыточным.

Отношения могут быть связаны различного рода связями. При связывании отношений необходимо определить основное (главное) и подчиненное отношение. Таким образом, в реляционной модели между отношениями поддерживается иерархическая связь. Имеются следующие варианты связей.

Связь “один-к-одному” (1:1). Одному кортежу главного отношения соответствует один кортеж подчиненного отношения. В этом случае фактически оба отношения являются равноправными.

Связь “один-ко-многим” (1:M). Одному кортежу основного отношения соответствует множество кортежей подчиненного отношения.

Связь “многие-к-одному” (M:1). Нескольким кортежам одного отношения соответствует единственный кортеж другого отношения.

Связь “многие-ко-многим” (M:M). Нескольким кортежам главного отношения соответствует множество кортежей подчиненного.

Для связи между двумя отношениями в каждом из них должен присутствовать набор атрибутов, по которым они связываются. В главном отношении это атрибуты, которые образуют первичный ключ. В подчиненном отношении в случае множественной связи должен быть набор атрибутов (внешний ключ), соответствующий первичному ключу основного отношения, и в подчиненном отношении он определяет не один, а множество кортежей, поскольку одному кортежу главного отношения может соответствовать несколько кортежей подчиненного отношения. Внешний ключ может быть, в принципе, реализован и в случае связи 1:1.

В реляционной модели на внешние ключи всегда накладывается ограничение, которое называется ссылочной целостностью. Под этим понимается, что каждому значению внешнего ключа всегда должны соответствовать кортежи в связываемых отношениях.

При проектировании реляционной модели БД необходимо контролировать целостность связей между отношениями. Для этого необходимо выполнение следующих правил.

1. Каждому кортежу главного отношения должно соответствовать ноль или более кортежей подчиненного отношения.

2. Каждому кортежу подчиненного отношения должен всегда соответствовать единственный кортеж в главном отношении – не может быть ситуации, когда кортеж подчиненного отношения не связан ни с одним кортежем главного отношения.

При проектировании реляционной БД одним из основных этапов является логическое проектирование, которое заключается прежде всего в определении количества отношений со своими схемами, установлении количества и типов связей между отношениями. Другими словами, на этом этапе определяется структуризация данных. Имеется несколько подходов структурирования данных, среди которых рассмотрим классический подход, который был исторически сформулирован первым и может быть определен следующим образом:

сбор информации об объектах реального мира, которые должны быть представлены в проектируемой БД, объединение этой информации в одной таблице и последующая декомпозиция (рис. 2.1) этой информации в несколько взаимосвязанных таблиц на основе метода нормальных форм (нормализации отношений).

Метод нормальных форм является классическим методом при проектировании реляционных БД. Процесс нормализации является итерационным: каждая итерация приводит отношение к нормальной форме более высокого уровня, которая обладает лучшими свойствами по сравнению с предыдущей нормальной формой.

Последовательность нормальных форм следующая:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);

- нормальная форма Бойса-Кодда или третья улучшенная нормальная форма(BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма или форма проекции-соединения (PJNF).

Рис. 2.1. Проектирование логической схемы БД

Важно еще раз подчеркнуть, что при нормализации отношений происходит их декомпозиция, т. е. после завершения этого процесса окончательный и исходный наборы отношений будут различными. Это означает, что исходная и окончательная схемы БД будут разными. Однако в основе теории нормализации лежит *принцип обратимости*, в соответствии с которым на любом этапе процесса нормализации исходная схема БД всегда может быть однозначно восстановлена.

Другими словами, процесс декомпозиции отношений всегда сохраняет эквивалентность схем БД на любом этапе.

Метод нормальных форм основывается на фундаментальном понятии зависимости между атрибутами отношений.

Говорят, что один атрибут функционально зависит от другого ($A \rightarrow B$, т. е. атрибут B зависит от атрибута A), если каждому значе-

нию независимого атрибута *A* однозначно соответствует определенное значение зависимого атрибута *B*. Частичной функциональной зависимостью называется зависимость неключевого атрибута от части составного первичного ключа. Если же неключевой атрибут зависит от всего составного первичного ключа, то такая зависимость называется полной.

Транзитивной зависимостью атрибута *C* от атрибута *A* называется такая зависимость, когда атрибут B функционально зависит от атрибута *A*, атрибут *C* зависит от B и при этом атрибут *B* не зависит от атрибута *C*.

Многозначной зависимостью атрибута *B* от атрибута *A* называется такая зависимость, когда данному значению атрибута *A* соответствует множество значений атрибута *B*, не зависящих от других атрибутов в данном отношении.

Взаимно независимыми атрибутами отношения называются такие атрибуты, когда ни один из них не является функционально зависимым от других атрибутов.

Исходное отношение

Преподаватель	Предмет	Группа
Петренко А.Л.	Линейная алгебра	12-01
	Аналитическая геометрия	11-02
Грушин К.К.	Общая физика	07-12
	Молекулярная динамика	12-03

Отношение в 1НФ

Преподаватель	Предмет	Группа
Петренко А.Л.	Линейная алгебра	12-01
Петренко А.Л.	Аналитическая геометрия	11-02
Грушин К.К.	Общая физика	07-12
Грушин К.К.	Молекулярная динамика	12-03

Рис. 2.2. Приведение реляционного отношения к первой нормальной форме

Перечисленные определения позволяют дать определения нормальным формам отношений.

Определение 1. Отношение находится в первой нормальной форме (1 NF), если в любом допустимом значении этого отношения каждый его кортеж содержит только одно значение для каждого атрибута. Другими словами, все атрибуты отношения должны принимать единственное значение.

На рис. 2.2 приведен пример приведения отношения к первой нормальной форме. Как видно, в исходном отношении в полях “Предмет” и “Группа” находятся не единичные значения атрибутов. После переведения отношения к 1 NF эта проблема разрешается.

Определение 2. Отношение находится во второй нормальной форме (2 NF), если оно находится в 1 NF и все его атрибуты, не входящие в первичный ключ (неключевые), связаны полной функциональной зависимостью с атрибутами первичного ключа.

Рис. 2.3. Приведение реляционного отношения
ко второй нормальной форме

Приведенное на рис. 2.3 исходное отношение отражает процесс сдачи студентами экзаменационной сессии. С учетом того, что студенты сдают несколько предметов, ясно, что первичный ключ в этом отношении является составным: для идентификации единст-

венного кортежа отношения необходимо задать значения номера зачетной книжки студента и названия предмета.

Исходное отношение не находится во второй нормальной форме, поскольку в нем имеются неполные функциональные зависимости неключевых атрибутов от атрибутов первичного ключа, а именно: неключевые атрибуты “ФИО” и “Группа” однозначно определяются значением только номера зачетной книжки. Самым простым способом приведения исходного отношения ко второй нормальной форме является декомпозиция его на 2 новых отношения, причем таким образом, чтобы в одном из них оказались как раз те неключевые атрибуты, которые в исходном отношении давали неполные функциональные зависимости. На рис. 2.3 представлены схемы этих новых отношений.

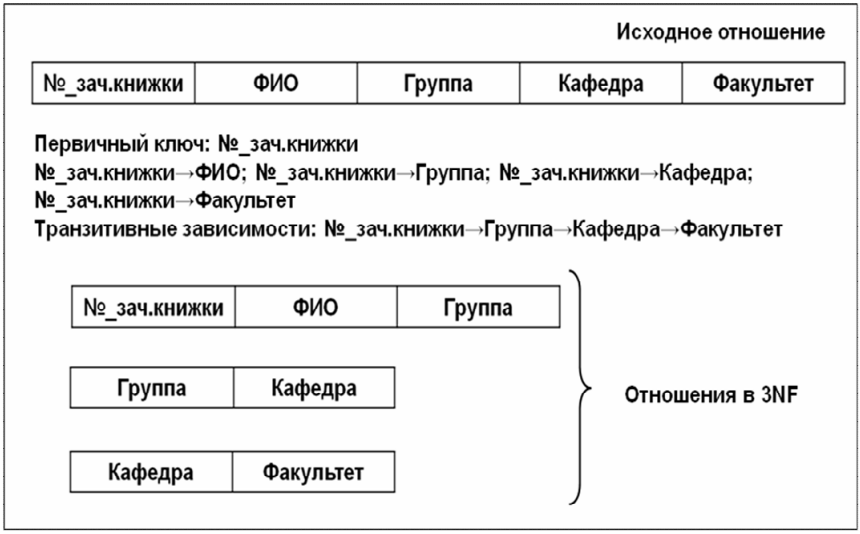


Рис. 2.4. Приведение реляционного отношения
к третьей нормальной форме

Определение 3. Отношение находится в третьей нормальной форме (3 NF), если оно находится в 2 NF и все его неключевые атрибуты взаимно независимы и полностью зависят от первичного ключа.

Нетрудно видеть, что в этом определении говорится о том, что для неключевых атрибутов не должно существовать транзитивных зависимостей от атрибутов первичного ключа.

Приведенное на рис. 2.4 отношение содержит информацию о том, в какой группе числится студент, к какой кафедре относится эта группа и на каком факультете расположена эта кафедра. Первичным ключом в этом отношении, очевидно, является атрибут "№ зачетной книжки". Ключ является простым, и отношение удовлетворяет определению второй нормальной формы. Однако отношение не находится в третьей нормальной форме, поскольку в нем имеется следующая транзитивная зависимость: №_зачетной книжки → Группа → Кафедра → Факультет. Приведение к третьей нормальной форме так же, как и в предыдущем случае, может быть произведено путем декомпозиции. Однако теперь уже требуется разбиение на 3 новых отношения, в каждом из которых транзитивные зависимости отсутствуют.

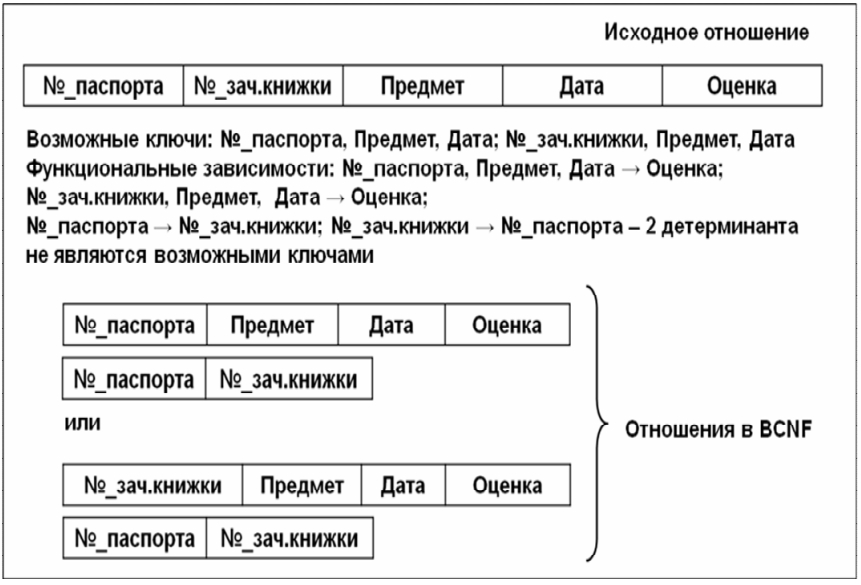


Рис. 2.5. Приведение реляционного отношения
к нормальной форме Бойса-Кодда

Определение 4. Отношение находится в нормальной форме Бойса-Кодда (BCNF), если оно находится в 3 NF и любая функциональная зависимость между ее атрибутами сводится к полной функциональной зависимости от возможного первичного ключа.

Это означает, что в отношении должны отсутствовать зависимости атрибутов составного первичного ключа от неключевых атрибутов: каждый детерминант отношения должен являться возможным первичным ключом.

Отношение, приведенное на рис. 2.5, содержит информацию о сдаче студентами экзаменационной сессии. Предположим, что эта информация является промежуточной в том смысле, что в таблицу заносятся данные как об успешных, так и неуспешных попытках студентов, для чего в этом отношении имеется атрибут “Дата” (экзамен по одному и тому же предмету студент может сдавать в несколько попыток, в один и тот же день можно попытаться сдать экзамены по нескольким предметам). Пусть, кроме того, в данное отношение наряду с номером зачетной книжки вносятся паспортные данные студентов.

Исходное отношение содержит, по крайней мере, два составных возможных ключа: “№ паспорта, Предмет, Дата” и “№ зачетной книжки, Предмет, Дата”. Атрибут “Оценка” функционально зависит от каждого из возможных ключей, эта зависимость является полной, неполных и транзитивных зависимостей в отношении нет. Поэтому данное отношение находится в третьей нормальной форме.

Однако отношение не удовлетворяет определению нормальной формы Бойса-Кодда. В самом деле, в нем имеются еще две функциональные зависимости: № паспорта → № зачетной книжки; № зачетной книжки → № паспорта. Это значит, что два детерминанта отношения не являются его возможными первичными ключами. Приведение отношения к BCNF производится путем декомпозиции таким образом, чтобы два детерминанта, не являющиеся возможными ключами (№ паспорта и № зачетной книжки), были в разных отношениях. В данном случае это может быть сделано двумя равноправными способами, как это показано на рис. 2.5.

Как правило, в подавляющем большинстве случаев процесс нормализации отношений бывает достаточно остановить на доведении отношения до нормальной формы Бойса-Кодда. Четвертая и тем боле пятая нормальные формы на практике используются редко. Для их определения необходимо ввести понятия нескольких операций реляционной алгебры, аппарат которой будет рассмотрен в следующем разделе. После этого вернемся к нормальным формам.

2.2. Реляционная алгебра, реляционное исчисление

Для работы с отношениями используется аппарат реляционной алгебры, предложенный Ф.Э. Коддом. Этот аппарат мог возникнуть потому, что в теории отношений доказывается, что их множество всегда замкнуто относительно некоторого набора операций, образуя таким образом вместе с этими операциями абстрактную алгебру. Это свойство и было использовано при разработке языка манипулирования данными, получившего название реляционной алгебры.

Операции реляционной алгебры Кодда делятся на основные теоретико-множественные и специальные реляционные. В первую группу операций входят классические операции теории множеств: объединение, пересечение, разность и расширенное декартово произведение. Вторая группа операций разработана непосредственно для решения задач манипулирования данными и в нее входят операции выборки (фильтрации, горизонтального выбора), проекции, деления и соединения (условного соединения или соединения по условию).

Рассмотрим сначала основные операции реляционной алгебры.

Объединением двух отношений R1 и R2, имеющих одинаковые схемы, называется отношение $R = R1 \cup R2$, множеством кортежей которого являются все кортежи исходных отношений.

Пример получения объединения отношений приведен на рис. 2.6.

Пересечением двух отношений R1 и R2, имеющих одинаковые схемы, называется отношение $R = R1 \cap R2$, множеством кортежей

которого являются кортежи одновременно первого и второго отношений.

Пример получения пересечения отношений приведен на рис. 2.7.

R1

Таб. №	ФИО	Предмет
21	Галкин В.А.	Биология
22	Палкин С.А.	Паскаль
27	Малкин Е.Д.	Физика
34	Скалкин Р.Р.	C++

R2

Таб. №	ФИО	Предмет
17	Реймер А.В.	Oracle
21	Галкин В.А.	Биология
25	Рахалин К.В.	Философия
34	Скалкин Р.Р.	C++
556	Брехунов Е.Е.	КСЕ

Объединение R1 и R2

Таб. №	ФИО	Предмет
17	Реймер А.В.	Oracle
21	Галкин В.А.	Биология
22	Палкин С.А.	Паскаль
25	Рахалин К.В.	Философия
27	Малкин Е.Д.	Физика
34	Скалкин Р.Р.	C++
556	Брехунов Е.Е.	КСЕ

Рис. 2.6. Объединение двух отношений

R1

Таб. №	ФИО	Предмет
21	Галкин В.А.	Биология
22	Палкин С.А.	Паскаль
27	Малкин Е.Д.	Физика
34	Скалкин Р.Р.	C++

R2

Таб. №	ФИО	Предмет
17	Реймер А.В.	Oracle
21	Галкин В.А.	Биология
25	Рахалин К.В.	Философия
34	Скалкин Р.Р.	C++
556	Брехунов Е.Е.	КСЕ

Пересечение R1 или R2

Таб. №	ФИО	Предмет
21	Галкин В.А.	Биология
34	Скалкин Р.Р.	C++

Рис. 2.7. Пересечение двух отношений

Разностью двух отношений R1 и R2, имеющих одинаковые схемы, называется отношение R = R1 \ R2, множеством кортежей которого являются кортежи, принадлежащие первому отношению и не принадлежащие второму.

В отличие от операций объединения и пересечения операция разности не является коммутативной.

Примеры получения разностей отношений приведены на рис. 2.8.

R1

Таб. №	ФИО	Предмет
21	Галкин В.А.	Биология
22	Палкин С.А.	Паскаль
27	Малкин Е.Д.	Физика
34	Скалкин Р.Р.	C++

Разность R1\R2

Таб. №	ФИО	Предмет
22	Палкин С.А.	Паскаль
27	Малкин Е.Д.	Физика

R2

Таб. №	ФИО	Предмет
17	Реймер А.В.	Oracle
21	Галкин В.А.	Биология
25	Рахалин К.В.	Философия
34	Скалкин Р.Р.	C++
556	Брехунов Е.Е.	КСЕ

Разность R2\R1

Таб. №	ФИО	Предмет
17	Реймер А.В.	Oracle
25	Рахалин К.В.	Философия
556	Брехунов Е.Е.	КСЕ

Рис. 2.8. Разности двух отношений

Рассмотренные операции реляционной алгебры применимы только к совместимым отношениям, т. е. к отношениям, имеющим эквивалентные схемы. В отличие от них, операция расширенного декартова произведения не накладывает никаких условий на схемы исходных отношений, поэтому она может быть выполнена над любыми двумя отношениями.

Для определения операции расширенного декартова произведения необходимо предварительно определить операцию сцепления двух кортежей.

R1

ФИО	Предмет
Галкин В.А.	Биология
Палкин С.А.	Паскаль
Малкин Е.Д.	Физика
Скалкин Р.Р.	C++

R2

Группа
Т7-25
Т9-37

Произведение R1XR2

ФИО	Предмет	Группа
Галкин В.А.	Биология	Т7-25
Галкин В.А.	Биология	Т9-37
Палкин С.А.	Паскаль	Т7-25
Палкин С.А.	Паскаль	Т9-37
Малкин Е.Д.	Физика	Т7-25
Малкин Е.Д.	Физика	Т9-37
Скалкин Р.Р.	C++	Т7-25
Скалкин Р.Р.	C++	Т9-37

Рис. 2.9. Произведение двух отношений

Сцеплением двух кортежей называется кортеж, который получается путем добавления атрибутов второго из сцепляемых кортежей непосредственно за последним атрибутом первого.

Расширенным декартовым произведением (или просто *произведением*) двух отношений R1 и R2 со степенями m и n соответственно называется отношение R = R1 x R2 степени $m + n$, кортежи которого получаются сцеплением кортежей первого отношения с кортежами второго.

Для того, чтобы получить произведение двух кортежей, необходимо, чтобы в них не было одинаковых имен атрибутов. Что же касается схем исходных отношений, то в отличие от рассмотренных ранее операций, они могут быть произвольными.

Пример применения операции произведения приведен на рис. 2.9.

Операция произведения может считаться симметричной в том смысле, что она допускает возможность перестановки атрибутов в

отношении. Достаточно часто эта операция используется для получения отношения, характеризующего все возможные комбинации между элементами отдельных множеств. Однако при этом сама по себе эта операция достаточно редко имеет самостоятельное значение. Результат, получаемый с ее помощью, является промежуточным и требует дальнейшей обработки.

R1

ФИО	Предмет	Группа
Галкин В.А.	Биология	Т7-25
Галкин В.А.	Биология	Т9-37
Палкин С.А.	Паскаль	Т7-25
Палкин С.А.	Паскаль	Т9-37
Малкин Е.Д.	Физика	Т7-25
Малкин Е.Д.	Физика	Т9-37
Скалкин Р.Р.	С++	Т7-25
Скалкин Р.Р.	С++	Т9-37

R2: Группа:='Т7-25'

ФИО	Предмет	Группа
Галкин В.А.	Биология	Т7-25
Палкин С.А.	Паскаль	Т7-25
Малкин Е.Д.	Физика	Т7-25
Скалкин Р.Р.	С++	Т7-25

R3: Предмет:='С++'

ФИО	Предмет	Группа
Скалкин Р.Р.	С++	Т7-25
Скалкин Р.Р.	С++	Т9-37

Рис. 2.10. Результаты применения операции выборки

Теперь дадим определения специальных операций.

Выборкой отношения по условию *w* является новое отношение с той же самой схемой, кортежи которого удовлетворяют условию *w*. В качестве условия выступают имена атрибутов, логические операции и операции сравнения, константы.

На рис. 2.10 в качестве условий выборки взяты значения атрибутов “Группа” и “Предмет”.

Проекцией отношения на подмножество его атрибутов является отношение, содержащие эти атрибуты и все кортежи исходного отношения, в которых содержатся значения данных атрибутов (рис. 2.11).

При операции проектирования запрещается указывать одинаковые атрибуты. Операция проектирования используется для получения требуемых характеристик объекта. Зачастую эта операция, так же как и операция расширенного декартова произведения, используется в качестве промежуточного шага (главным образом, в опе-

рациях выборки). Самостоятельное значение эта операция имеет, в основном, на заключительном этапе вычислений для получения результата запроса.

R1

ФИО	Предмет	Группа
Галкин В.А.	Биология	Т7-25
Галкин В.А.	Биология	Т9-37
Палкин С.А.	Паскаль	Т7-25
Палкин С.А.	Паскаль	Т9-37
Малкин Е.Д.	Физика	Т7-25
Малкин Е.Д.	Физика	Т9-37
Скалкин Р.Р.	С++	Т7-25
Скалкин Р.Р.	С++	Т9-37

R2: R1[ФИО, Группа]

ФИО	Группа
Галкин В.А.	Т7-25
Галкин В.А.	Т9-37
Палкин С.А.	Т7-25
Палкин С.А.	Т9-37
Малкин Е.Д.	Т7-25
Малкин Е.Д.	Т9-37
Скалкин Р.Р.	Т7-25
Скалкин Р.Р.	Т9-37

Рис. 2.11. Результат применения операции проектирования

R1

ФИО	Должность	Предмет
Галкин В.А.	Доцент	Биология
Палкин С.А.	Профессор	Паскаль
Малкин Е.Д.	Ассистент	Физика
Скалкин Р.Р.	Профессор	С++

R2

Должность	Оклад
Профессор	5000
Доцент	3000
Ассистент	2000

R3

ФИО	Должность	Предмет	Оклад
Галкин В.А.	Доцент	Биология	3000
Палкин С.А.	Профессор	Паскаль	5000
Малкин Е.Д.	Ассистент	Физика	2000
Скалкин Р.Р.	Профессор	С++	5000

Рис. 2.12. Соединение двух отношений

Частными случаями операции проекции являются операция тождественной проекции (результатом ее выполнения является исходное отношение) и операция пустой проекции (ее результатом является пустое множество).

Операция *соединения* отношений выполняется над двумя отношениями. Соединением двух отношений по условию *w* является отношение, получаемое путем декартова произведения данных отношений с последующим применением к полученному результату операции выборки по условию *w*.

На рис.2.12, на котором иллюстрируется эта операция, в качестве условия соединения является атрибут “Должность”, общий для двух исходных отношений.

Операцию деления можно определить следующим образом. Пусть имеются два отношения R1 и R2, *A* – подмножество множества атрибутов отношения R1, *B* – подмножество множества атрибутов обоих отношений.

Делением отношения R1 на отношение R2 по общему подмножеству атрибутов *B* является отношение, состоящее из множества атрибутов *A* и содержащее набор таких кортежей *a*, что в отношении R1 имеются наборы кортежей *a* и *b*, где набор кортежей *b* является множеством значений подмножества *B* отношения R2.

Операция деления проиллюстрирована на рис. 2.13.

Операция деления используется в основном тогда, когда необходимо провести сравнение некоторого множества характеристик атрибутов отношений. Вообще говоря, эта операция является достаточно сложной для ее абстрактного представления и она всегда может быть заменена сочетанием других операций реляционной алгебры.

Необходимо отметить, что предложенный Коддом набор операций реляционной алгебры имеет ряд недостатков. Действительно, эти операции являются избыточными, поскольку минимальный необходимый набор составляют пять операций – объединение, разность, произведение, выборка и проекция. Остальные три операции можно определить через них.

Кроме того, операций реляционной алгебры недостаточно для построения реальной СУБД. Поэтому Дейтом были предложены дополнительные операции – переименования, расширения, подве-

дения итогов, присвоения, вставки, обновления, удаления, сравнения.

R1

A	B
Значение A1	Значение B1
Значение A1	Значение B2
Значение A1	Значение B3
Значение A1	Значение B4
Значение A1	Значение B5
Значение A1	Значение B6
Значение A2	Значение B1
Значение A2	Значение B2
Значение A3	Значение B2
Значение A4	Значение B2
Значение A4	Значение B4
Значение A5	Значение B5

R2

B
Значение B2
Значение B4

R1:R2

A
Значение A1
Значение A4

Рис. 2.13. Операция

Операция *переименования* предназначена для изменения имени атрибута отношения. Ее расширенным вариантом является операция множественного переименования, которая позволяет одновременно изменять имена нескольких атрибутов в отношении.

Применение операции *расширения* приводит к возникновению нового отношения, содержащего полный набор атрибутов исходного отношения плюс новый дополнительный атрибут, значения которого получаются путем некоторых вычислений. Как частный случай, операция расширения может использоваться и для переименования атрибута. Операция множественного расширения позволяет одновременно добавлять в исходное отношение несколько новых вычисляемых атрибутов.

Операция *подведения итогов* используется для выполнения неких групповых вычислений. Ее результатом является отношение, содержащее список части атрибутов исходного отношения, а также новый атрибут, который вычисляется. Таким образом, при выполнении операции подведения итогов фактически выполняется проецирование исходного отношения на часть атрибутов, а затем каждый кортеж проекции дополняется новым вычисляемым атрибутом. Так же, как и в предыдущих случаях, данная операция дополняется операцией множественного подведения итогов.

Операции *присвоения*, *вставки*, *обновления* и *удаления* предназначены для изменения тела исходного отношения. С помощью этих операций можно удалять и добавлять атрибуты и кортежи в исходном отношении, заменять значения атрибутов на новые, присваивать атрибутам вычисляемые значения

Операция *реляционного сравнения* используется для сравнения значений атрибутов в двух отношениях.

Все перечисленные операции обладают одним общим свойством – свойством замкнутости, которое заключается в том, что результатом выполнения любой из этих операций является отношение, которое далее может вновь участвовать в любой другой реляционной операции.

И, наконец, при использовании операций реляционной алгебры необходимо придерживаться следующих правил:

1. Так же, как и в обычной алгебре, в реляционной алгебре определен приоритет выполнения операций, который всегда нужно учитывать при записи выражений. При необходимости изменения порядка производимых вычислений используются скобки.

2. Одно и то же выражение может быть записано с помощью различных комбинаций операций реляционной алгебры. Это позволяет зачастую упрощать выражения.

3. Операции реляционной алгебры могут производиться как над одним, так и над двумя отношениями. В первом случае они называются *унарными*, во втором – *бинарными*. При выполнении бинарных операций реляционной алгебры следует помнить о совместимости исходных отношений: при выполнении действий над двумя отношениями необходимо следить за эквивалентностью их схем, степеней и в нужных случаях перед выполнением той либо иной операции совмещать исходные отношения (в частности, менять названия атрибутов). Отметим, что это ограничение действует не для всех операций: так, оно не применяется для бинарных операций декартова произведения и соединения

В заключение отметим, что в реляционной алгебре все действия выполняются над отношениями, и результатом всех выполненных операций также являются отношения. В этом контексте любой формализованный язык реляционной алгебры (в качестве примера можно привести язык запросов ISBL – Information System Base

Language) следует считать процедурным – искомое отношение вычисляется при выполнении последовательности реляционных операторов. Эти операторы состоят из операндов (отношений) и самих реляционных операций.

В отличие от языков реляционной алгебры существуют языки реляционного исчисления, которые являются непроцедурными и используются для построения запросов к БД, содержащих в себе лишь информацию о желаемом результате. Другими словами, вместо записи в явном виде набора операций, необходимого для получения искомого результата, указывается только необходимый набор свойств этого результата без конкретизации процедуры его получения. К языкам этого типа прежде всего относится SQL.

Рис. 2.14. Пример проецирования без потерь

Теперь в заключение раздела вернемся к теории нормальных форм и сформулируем определения четвертой и пятой нормальных форм.

Для определения четвертой нормальной формы необходимо предварительно сформулировать теорему Фейджина.

Теорема Фейджина. Отношение R с атрибутами *A*, *B* и *C* может быть без потерь спроецировано в отношения R1 с атрибутами *A* и *B* и R2 с атрибутами *A* и *C* только в том случае, когда в исходном отношении существуют две многозначные зависимости атрибутов *B* и *C* от атрибута *A*.

Пример проецирования без потерь исходного отношения R в отношения R1 и R2 проиллюстрирован на рис. 2.14.

В этой теореме проецирование без потерь означает, что при декомпозиции исходного отношения оно может быть восстановлено путем естественного соединения полученных отношений.

Рис. 2.15. Приведение реляционного отношения к четвертой нормальной форме

Смысл этой теоремы заключается в том, что в ней доказывается эквивалентность схем БД до проецирования и после при условии, что в исходном отношении имеется несколько многозначных зависимостей.

Определение 5. Отношение находится в четвертой нормальной форме (4 NF) тогда и только тогда, когда в случае существования многозначной зависимости атрибута *B* от атрибута *A* все остальные атрибуты этого отношения функционально зависят от атрибута *A*.

На рис. 2.15 приведено исходное отношение, содержащее 3 атрибута. В этом отношении имеется 2 многозначные зависимости атрибутов “Предмет” и “№ зачетной книжки” от атрибута “Груп-

па”. Для приведения этого отношения к четвертой нормальной форме необходимо декомпозировать его на два отношения таким образом, чтобы атрибуты “Предмет” и “№ зачетной книжки” оказались в разных отношениях, каждое из которых будет удовлетворять определению четвертой нормальной формы.

Исходное отношение

Преподаватель	Кафедра	Предмет
Глотов В.В.	Радиофизика	Общая физика
Козлов М.К.	Автоматика	Электротехника
Михеев С.В.	Уфология	Электротехника
Орлов М.У.	Автоматика	Колебания и волны
Соколов Б.Р.	Лазерная физика	Газовые лазеры
Соколов Б.Р.	Лазерная физика	Электротехника
Ткачук С.В.	Уфология	Маркетинг
Зотов Г.В.	Высшая математика	Мат. анализ
Белоус Ц.Д.	Высшая математика	Диф. уравнения
Шевченко П.П.	Химия	Мат. анализ

Рис. 2.16. Таблица базы данных

Для определения пятой нормальной формы необходимо ввести понятие *зависимости проекции-соединения* (*project-join* зависимости).

Отношение R с атрибутами *X*, *Y*, ..., *Z* удовлетворяет зависимости соединения (*X*, *Y*, ..., *Z*) тогда, когда оно может быть восстановлено без потерь путем соединения своих проекций на *X*, *Y*, ..., *Z*.

Определение 6. Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения PJ/NF) тогда и только тогда, когда любая зависимость соединения в этом отношении следует из существования в нем некоторого возможного ключа.

В качестве примера рассмотрим отношение, приведенное на рис. 2.16. Если предположить, что один и тот же преподаватель может совмещать работу сразу на нескольких кафедрах и, кроме того, вести занятия по нескольким предметам, то первичным клю-

чом этого отношения является набор всех его атрибутов. Поэтому данное отношение удовлетворяет определению четвертой нормальной формы.

R2={Преподаватель, Кафедра}

Преподаватель	Кафедра
Глотов В.В.	Радиофизика
Козлов М.К.	Автоматика
Михеев С.В.	Уфология
Орлов М.У.	Автоматика
Соколов Б.Р.	Лазерная физика
Соколов Б.Р.	Лазерная физика
Ткачук С.В.	Уфология
Зотов Г.В.	Высшая математика
Белоус Ц.Д.	Высшая математика
Шевченко П.П.	Химия

R3={Преподаватель, Предмет}

Преподаватель	Предмет
Глотов В.В.	Общая физика
Козлов М.К.	Электротехника
Михеев С.В.	Электротехника
Орлов М.У.	Колебания и волны
Соколов Б.Р.	Газовые лазеры
Соколов Б.Р.	Электротехника
Ткачук С.В.	Маркетинг
Зотов Г.В.	Мат. анализ
Белоус Ц.Д.	Диф. уравнения
Шевченко П.П.	Мат. анализ

R4={Кафедра, Предмет}

Кафедра	Предмет
Автоматика	Электротехника
Автоматика	Колебания и волны
Высшая математика	Мат. анализ
Высшая математика	Диф. уравнения
Лазерная физика	Газовые лазеры
Лазерная физика	Электротехника
Радиофизика	Общая физика
Уфология	Электротехника
Уфология	Маркетинг
Химия	Мат. анализ

Рис. 2.17. Декомпозиция исходного отношения

Если исходное отношение R1 удовлетворяет зависимости проекции соединения (Преподаватель, Кафедра, Предмет), то оно не находится в пятой нормальной форме, поскольку наличие зависимости проекции-соединения определяется наборами атрибутов, не являющихся возможными ключами.

(R3, R4)

Преподаватель	Кафедра	Предмет
Глотов В.В.	Радиофизика	Общая физика
Козлов М.К.	Автоматика	Электротехника
Козлов М.К.	Лаз. физ.	Электротехника
Козлов М.К.	Уфология	Электротехника
Михеев С.В.	Автоматика	Электротехника
Михеев С.В.	Лаз. физ.	Электротехника
Михеев С.В.	Уфология	Электротехника
Орлов М.У.	Автоматика	Кол. и волны
Соколов Б.Р.	Лаз. физ.	Газ. лазеры
Соколов Б.Р.	Автоматика	Электротехника
Соколов Б.Р.	Лаз. физ.	Электротехника
Соколов Б.Р.	Уфология	Электротехника
Ткачук С.В.	Уфология	Электротехника
Ткачук С.В.	Уфология	Маркетинг
Зотов Г.В.	Высш.матем.	Мат. анализ
Зотов Г.В.	Высш.матем.	Электротехника
Белоус Ц.Д.	Высш.матем.	Мат. анализ
Белоус Ц.Д.	Высш.матем.	Электротехника
Шевченко П.П.	Химия	Мат. анализ

Рис. 2.18. Соединение отношений R3 и R4

Произведем декомпозицию этого отношения в три новых отношения таким образом, как это показано на рис. 2.17. Отношения R2, R3 и R4 находятся в пятой нормальной форме, что можно проверить, получив все попарные соединения (R2, R3), (R2, R4) и (R3, R4) так, как это показано на рис. 2.18.

Еще раз укажем, что на практике отношения обычно нормализуются до 3 NF или BCNF. Сам процесс нормализации обычно заключается в последовательном выполнении следующих действий.

1. Удаляются частичные функциональные зависимости неключевых атрибутов от атрибутов первичного ключа.

2. Удаляются транзитивные зависимости неключевых атрибутов от атрибутов первичного ключа.

3. Удаляются зависимости атрибутов составных ключей от неключевых атрибутов.

Таким образом, процесс нормализации отношений осуществляется пошагово и заключается в последовательном переводе отношения от первой нормальной формы к нормальным формам более высокого порядка. При этом каждая следующая нормальная форма сохраняет все свойства предыдущих.

Контрольные вопросы

1. Что называется отношением?
2. Что такое реляционная модель данных?
3. Перечислите теоретико-множественные и специальные операции реляционной алгебры.
4. Дайте определения операций объединения, пересечения и разности отношений.
5. Дайте определение операции расширенного декартова произведения.
6. Что называется схемой отношения? В каком случае отношения имеют эквивалентные схемы?
7. Что называется степенью отношения?
8. В чем заключается операция сцепления кортежей двух отношений?
9. Дайте определения операций выборки, проектирования, соединения и деления отношений.
10. В каких случаях, как правило, применяется операция деления отношений?
11. Перечислите дополнительные операции реляционной алгебры Дейта.
12. В чем заключается свойство замкнутости дополнительных реляционных операций?
13. Какие операции реляционной алгебры называются унарными, а какие – бинарными?
14. Что означает, что формализованный язык реляционной алгебры является процедурным? Какие языки является непроцедурными и для каких целей они используются?

15. Какие объекты в реляционной алгебре называются доменами?

16. Что такое полное декартово произведение?

17. Какие условия должны быть выполнены для того, чтобы двумерная таблица могла быть интерпретирована как отношение?

18. Что называется первичным ключом отношения? Какие ключи являются простыми, а какие – составными?

19. Может ли отношение не иметь первичного ключа?

20. Что называется возможным ключом отношения?

21. Какие связи могут существовать между отношениями? Как называются наборы атрибутов, по которым осуществляются эти связи?

22. Что такое ссылочная целостность? В чем заключается принцип поддержания целостности?

23. В чем заключается метод нормальных форм?

24. Дайте определение функциональной зависимости атрибутов отношения.

25. Какие зависимости называются полными, а какие – неполными?

26. Какие функциональные зависимости называются транзитивными и многозначными?

27. Дайте определения всех нормальных форм. Приведите примеры отношений, находящихся в какой-либо из существующих нормальных форм.

28. Сформулируйте теорему Фейджина.

29. Дайте определение зависимости проекции-соединения.

30. Сформулируйте алгоритм нормализации отношений.

3. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

3.1. Концептуальное моделирование

Концептуальное проектирование является вторым этапом проектирования БД, который следует непосредственно после анализа предметной области. Эта стадия заключается в формализованном описании предметной области, которое должно быть таким, чтобы, с одной стороны, можно было проанализировать корректность схемы разрабатываемого проекта БД, а с другой стороны, оно не было привязано к конкретной СУБД, выбор которой будет произведен позднее. Такое формализованное описание предметной областью будем называть *концептуальной моделью* БД.

Основными свойствами, которым должна удовлетворять концептуальная модель БД, являются прежде всего следующие.

1. Предметная область должна быть описана адекватно и непротиворечиво.

2. Все возможные пользователи должны трактовать эту модель однозначно и легко ее воспринимать.

3. При необходимости концептуальная модель должна иметь возможность легко модифицироваться, подвержена композиции и декомпозиции.

При описании предметной области используются формализованные языковые средства, среди которых чаще всего используются графические. Вообще говоря, для описания предметной области можно было бы использовать и естественный язык, однако большим недостатком при таком подходе является громоздкость и неоднозначность интерпретации.

Первым шагом при разработке концептуальной модели БД является описание объектов предметной области и связей между ними. Концептуальное моделирование направлено на отображение семантики (то есть смысла) предметной области на модель БД. На ранних этапах развития теории БД было предложено несколько семантических моделей. Все эти модели имели свои преимущества и недостатки и к настоящему моменту времени в подавляющем большинстве случаев используется так называемая модель “сущность-связь”. Эта модель называется также методом *ER*-диаграмм

или *ER*-моделей (*Entity-Relationship Model*). Рассмотрим этот метод более подробно (рис. 3.1).

Рис. 3.1. Основные понятия модели “сущность-связь”

Основными понятиями этого метода являются *сущность* и *связь между сущностями*. *Сущностью* называется объект или класс однотипных объектов, информация о которых содержится в БД. Каждая сущность имеет свое уникальное имя. Характеристики, определяющие свойства сущности, называются ее *атрибутами*, что эквивалентно понятию атрибута в отношении. В БД имеется множество *экземпляров* данной сущности, каждый из которых уникален и может быть однозначно идентифицирован. Для этого используется *ключ сущности* – набор ее атрибутов, однозначно определяющих экземпляр сущности. Таким образом, понятие ключа сущности аналогично понятию ключа отношения, а с учетом изложенного, сама сущность может быть интерпретирована как отношение.

Зависимость между сущностями определяет *связь* между ними. Связи делятся на три типа по *степени связи* сущностей: один-к-одному (1:1), один-ко-многим (или многие-к-одному) (1:М, М:1), многие-ко-многим (М:М). В общем случае между двумя сущностями может быть задано произвольное количество связей с различными смысловыми нагрузками. *Класс принадлежности* сущности может быть *обязательным* (О), когда в связи должен участвовать каждый экземпляр сущности, и *необязательным* (Н), когда не все экземпляры сущности должны участвовать в связи. Связь может быть обязательной со стороны одной сущности и необязательной со стороны другой. На практике степень связи и класс ее принадлежности всегда определяются, исходя из анализа предметной области.

В *ER*-моделях поддерживается принцип категоризации сущностей. Вводятся понятия супертипа и подтипа. Сущность, на основе которой строятся подтипы, называется супертипом и может быть представлена в виде двух или более своих подтипов – сущностей, каждая из которых имеет как общие свойства, которые могут наследоваться, так и уникальный для данного подтипа набор свойств. Любой экземпляр супертипа должен относиться к конкретному подтипу.

Для удобства проектирования концептуальной модели и повышения ее наглядности и легкости восприятия пользователями сущности, экземпляры сущностей и связи между ними представляются графически в виде *диаграмм ER-экземпляров*, *диаграмм ER-типов* (рис. 3.2) и *ER-диаграмм*. Представленная в виде таких диаграмм модель БД называется *ER*-моделью.

Таким образом, *ER*-модель является графическим описанием предметной области в терминах “сущность-свойство-связь” и представляет собой один из наиболее важных элементов концептуальной модели БД. Использование *ER*-моделирования прежде всего является удобным средством документирования проектируемой БД, не привязанным к какой-либо конкретной СУБД, что важно, поскольку, с одной стороны, выбор СУБД может быть произведен на более поздних этапах, а с другой стороны, при необходимости выбора другой СУБД не нужно заново проектировать модель БД.

Существует большое количество различных нотаций для построения *ER*-модели. В этих нотациях используются разные символы, линии и фигуры для указания на характеристики объектов БД и связей между ними. Так, в частности, для определения сущности используется, как правило, прямоугольный блок. Под блоком сущности указывается ее ключ, для чего часто используется подчеркивание ключевых атрибутов сущности. Обязательное участие сущности в связи может быть указано блоком с точкой внутри, смежным с блоком этой сущности. При необязательности связи дополнительный блок к блоку сущности не добавляется, а точка располагается на линии связи. Кроме того, на линии связи расставляются символы, указывающие на ее степень.

Некоторые примеры, относящиеся к одной нотации, приведены на рис. 3.2 – 3.5. На рис. 3.6 приведены примеры изображения связей для различных нотаций.

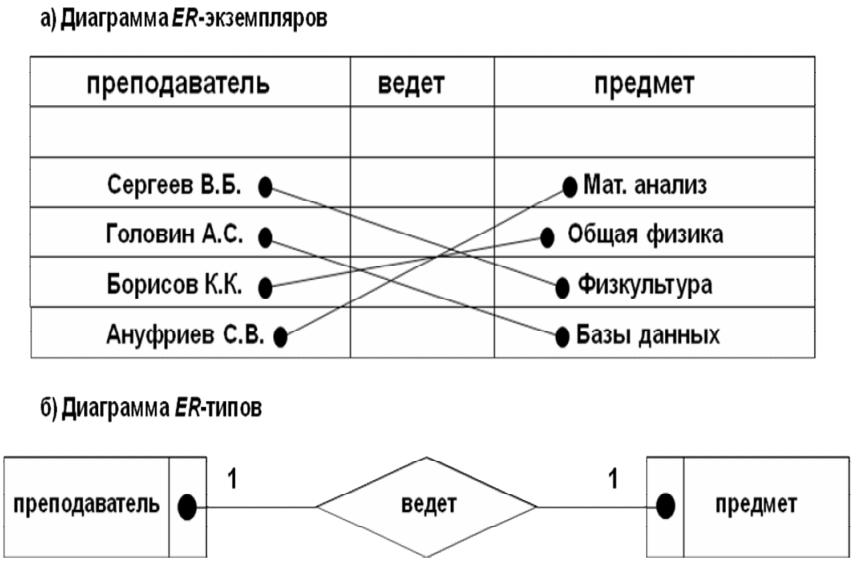
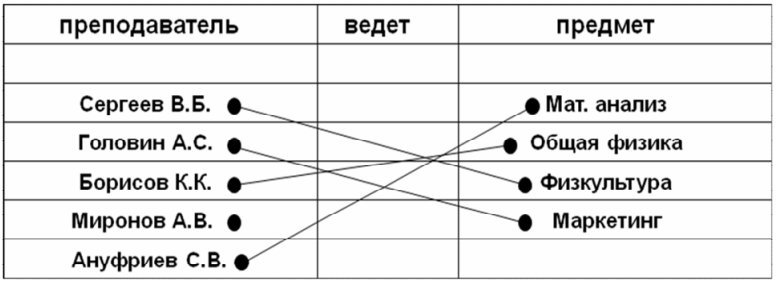


Рис. 3.2. Диаграммы *ER*-экземпляров и *ER*-типов для случая связи 1:1 с обязательными классами принадлежности обеих сущностей

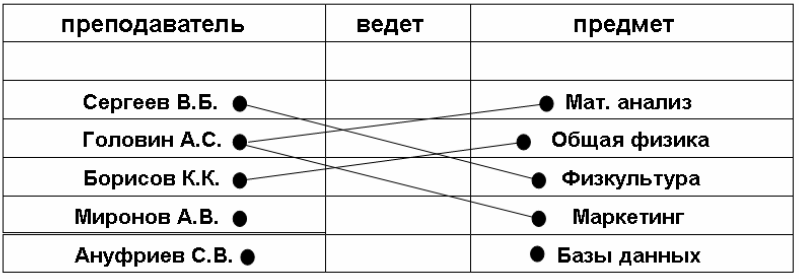
а) Диаграмма *ER*-экземпляров



б) Диаграмма *ER*-типов

Рис. 3.3. Диаграммы *ER*-экземпляров и *ER*-типов
для случая связи 1:1 с необязательным
классом принадлежности одной из сущностей

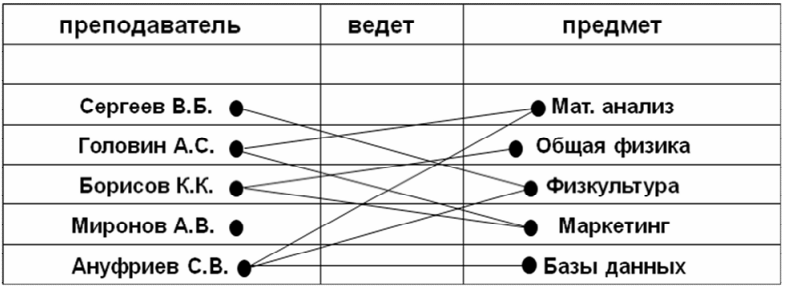
а) Диаграмма *ER*-экземпляров



б) Диаграмма *ER*-типов

Рис. 3.4. Диаграммы *ER*-экземпляров и *ER*-типов
для случая связи 1:M с необязательными
классами принадлежности обеих сущностей

а) Диаграмма *ER*-экземпляров



б) Диаграмма *ER*-типов

Рис. 3.5. Диаграммы *ER*-экземпляров и *ER*-типов
для случая связи M:M с необязательным
классом принадлежности первой сущности

Рис. 3.6. Примеры изображения связей между
сущностями в различных нотациях

Как уже говорилось, под каждым объектом предметной области понимается некая сущность. При построении *ER*-модели желательно давать каждой сущности словесную интерпретацию исходя из свойств объекта в контексте его принадлежности к проектируемой БД.

Объекты могут быть *реальными* или *абстрактными*. Объекты объединяются в *классы объектов* – совокупность объектов, имеющих одинаковый набор свойств. При этом *ER*-модель строится не для отдельных экземпляров объектов, а именно для классов объектов. В *ER*-модели каждому классу присваивается уникальное имя – существительное в единственном числе, дополненное при необходимости прилагательным или предлогом (ПРЕДМЕТ, ПРЕДМЕТ_ИЗУЧАЕМЫЙ). Уникальное имя объекта называется его *идентификатором*.

Каждый объект обладает набором *свойств*. В совокупности эти свойства описывают все возможные состояния объекта. Экземпляры объекта различаются как раз тем, что конкретные значения их свойств отличаются (объект СТУДЕНТ, свойства ФИО, Группа).

Если каждый объект может обладать только одним значением данного свойства в каждый момент времени (свойство Дата_Рождения), то такие свойства называются *единичными*. Если же одновременно у объекта имеется несколько значений свойства (свойство Изучаемый_Предмет у объекта СТУДЕНТ), то такие свойства называются *множественными*.

Если значения некоторых свойств не могут изменяться во времени, то такие свойства называются *статическими*, а если могут, то *динамическими*.

Есть такие свойства, которые могут одновременно присутствовать не у всех объектов данного класса (например, свойство Ученая_Степень у объекта ПРЕПОДАВАТЕЛЬ). Такие свойства называются *условными*.

Иногда вводится понятие *составного* свойства. Например, свойство Адрес может состоять из составных частей Город, Улица, Дом и т. д.

Существует несколько разновидностей объектов. *Простым* называется объект, если в данной конкретной модели БД он рассматривается как неделимый. *Сложным* объектом является такой объ-

ект, который рассматривается как объединение нескольких объектов. Сложные объекты в свою очередь делятся на составные, обобщенные и агрегированные.

Составной объект используется для отображения “целое-часть” (ГРУППА-СТУДЕНТ).

Обобщенный объект используется для отображения связи “род-вид” между разными объектами предметной области. Например, для некоторого предприятия обобщенный объект СОТРУДНИК (родовой объект) составляют объекты ДИРЕКТОР, БУХГАЛТЕР. МЕНЕДЖЕР и т. д., (видовые объекты) которые называются *категориями* обобщенного объекта). Здесь впервые встречается понятие *наследования* свойств, поскольку все видовые объекты должны обладать всеми свойствами родового объекта (наследуют его свойства), а также индивидуальными, присущими только данному видовому объекту свойствами.

Агрегированными называются такие объекты, которым соответствует некий процесс. Примером такого объекта может быть объект СЕССИЯ, который объединяет в себе такие объекты, как ДАТА_ЭКЗАМЕНА, ОЦЕНКА, ПРЕДМЕТ, СТУДЕНТ.

Некоторые примеры изображения различных объектов, обладающих разными свойствами, приведены на рис. 3.7 – 3.9.

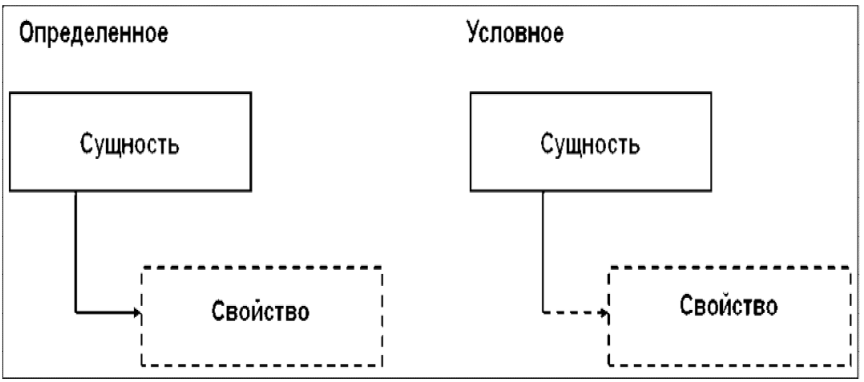


Рис. 3.7. Объекты с простым единичным свойством

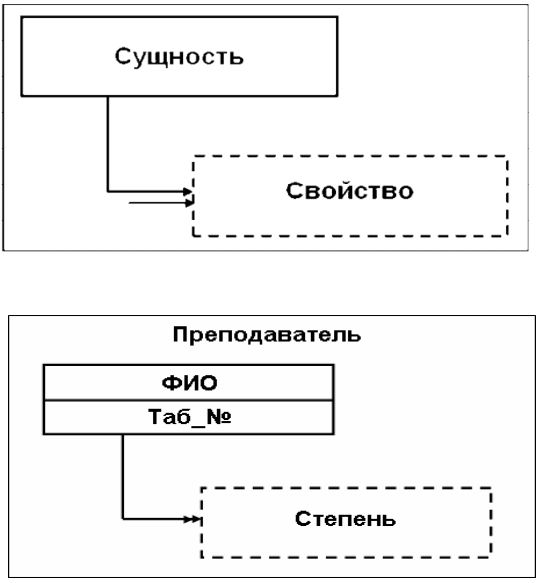


Рис. 3.8. Объект с простым множественным свойством

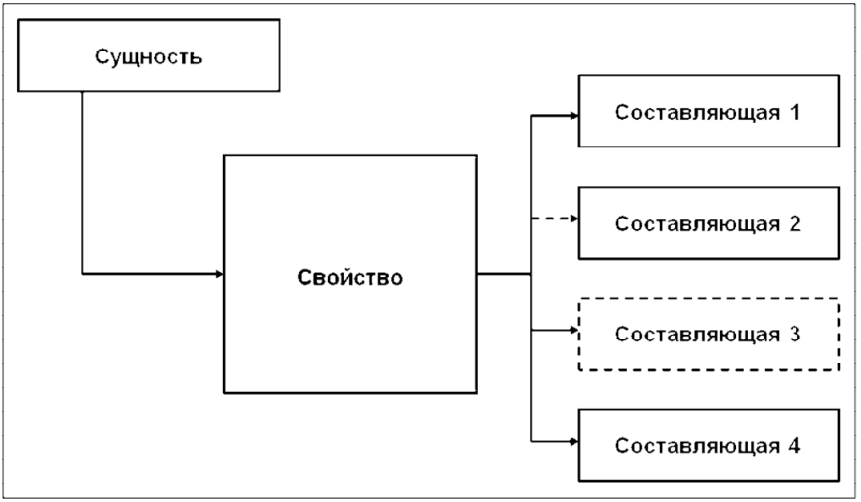


Рис. 3.9. Объект с составным свойством

Следующим этапом вслед за построением *ER*-модели является формирование отношений проектируемой БД. Вообще говоря, переход к отношениям можно формально отнести к следующему этапу проектирования БД – логическому проектированию, однако процесс проектирования БД является итерационным – зачастую происходит возврат к предыдущим этапам для пересмотра принятых ранее решений. Поэтому при формировании отношений иногда приходится изменять уже спроектированную *ER*-диаграмму в зависимости от возникающих обстоятельств.

Существуют вполне определенные правила в зависимости от степени связи между сущностями и класса принадлежности экземпляров сущностей. Сформулируем эти правила для всех возможных случаев.

Первый случай. Связь 1:1, класс принадлежности обеих сущностей обязательный. В этом случае для обеих сущностей (С1 и С2) формируется одно отношение *R*1. В этом отношении в качестве первичного ключа выбирается ключ любой из сущностей (К1 или К2), а атрибуты сущностей становятся атрибутами этого отношения. Сформированное таким образом отношение содержит в себе полную информацию об обеих сущностях (рис. 3.10).

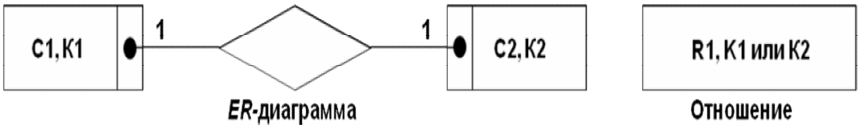


Рис. 3.10. Формирование реляционного отношения для случая 1:1, О-О

Второй случай. Связь 1:1, класс принадлежности одной сущности обязательный, а другой – необязательный (рис. 3.11). Для каждой сущности формируется отношение с соответствующим набором атрибутов, первичным ключом каждого из этих отношений является ключ соответствующей сущности. После этого для установления связи между отношениями к отношению, соответствующему сущности с обязательным классом принадлежности, добавляется атрибут, являющийся ключом сущности с необязательным классом принадлежности.

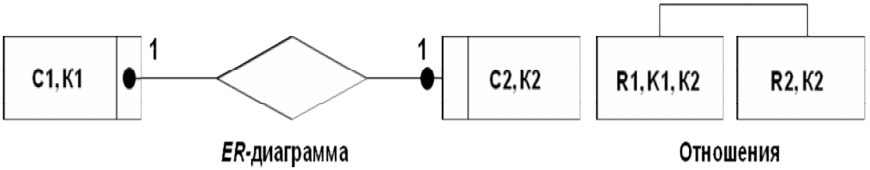


Рис. 3.11. Формирование реляционных отношений для случая 1:1, О-Н

Третий случай. Связь 1:1, классы принадлежности обеих сущностей необязательные (рис. 3.12). В таком случае создается три отношения. Первые два отношения соответствуют каждой из сущностей, их первичными ключами являются ключи этих сущностей. Третье отношение используется для связи между первыми двумя, и его атрибутами являются первичные ключи двух отношений.

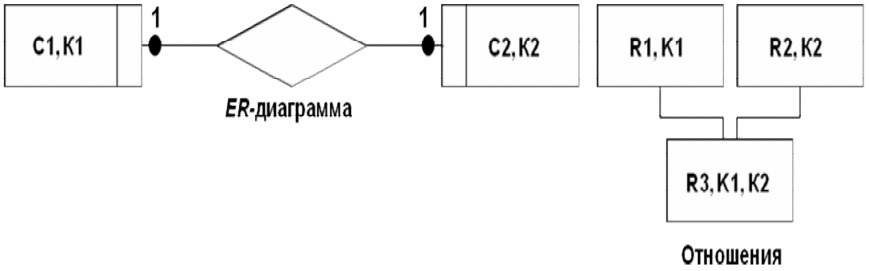


Рис. 3.12. Формирование реляционных отношений для случая 1:1, Н-Н

Четвертый случай. Связь 1:M (M:1), класс принадлежности M-связной сущности обязательный (рис. 3.13). Достаточно сформировать два отношения для каждой из сущностей так же, как и в случае 2. После этого ключ 1-связной сущности добавляется в качестве атрибута в отношение, соответствующее M-связной сущности, в качестве внешнего ключа.

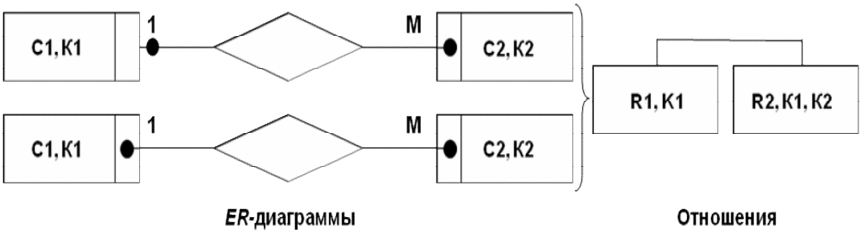


Рис. 3.13. Формирование реляционных отношений для случаев 1:M, Н-О и 1:M, О-О

Пятый случай. Связь 1:M (M:1), класс принадлежности M-связной сущности необязательный (рис. 3.14). Здесь по тому же принципу, как и в случае 3, необходимо сформировать три отношения.

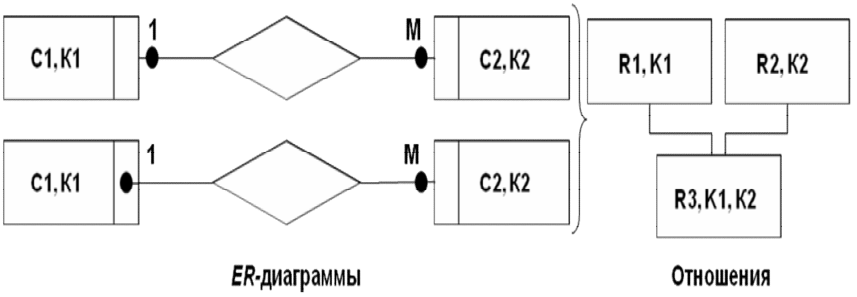


Рис. 3.14. Формирование реляционных отношений для случаев 1:M, H-H и 1:M, O-H

Необходимо отметить, что в случае связи один-ко-многим (многие-к-одному) правила формирования отношений не зависят от класса принадлежности односвязной сущности.

Шестой случай. Связь M:M (рис. 3.15). Вне зависимости от классов принадлежности каждой из сущностей формируются три отношения таким же образом, как и в случаях 3 и 5.

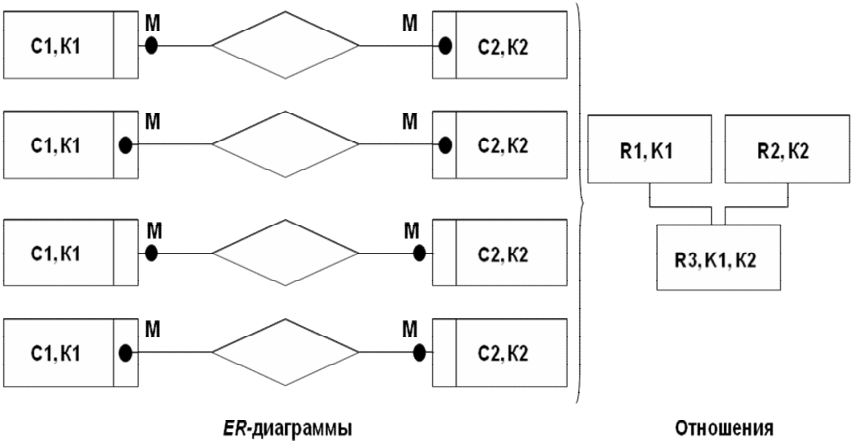


Рис. 3.15. Формирование реляционных отношений для случая M:M

В настоящее время существует целый ряд автоматизированных средств проектирования *ER*-моделей, называемых CASE-средствами. Среди них следует отметить Design/IDEF, Power Designer (S-Designor), Oracle, ERWin. Использование этих средств дает ряд преимуществ, а именно: снижаются требования к знанию языков описания данных и конкретных СУБД; автоматически контролируются целостность и согласованность схемы описания БД; сокращается в целом время проектирования; появляется возможность автоматического тестирования проекта на разных стадиях его проектирования.

Подводя итоги, сформулируем основные этапы проектирования БД в рамках концептуального моделирования.

1. Определение сущностей и выявление связей между ними.
2. Построение *ER*-диаграммы с учетом всех сущностей и связей между ними.
3. Формирование набора предварительных отношений с указанием предполагаемых первичных и внешних ключей.
4. Добавление неключевых атрибутов в отношения.
5. Нормализация предварительных отношений, приведение их по возможности к нормальной форме Бойса-Кодда. На этом этапе, как правило, и происходит пересмотр принятых решений и происходит возврат к предыдущим этапам проектирования, включающим пересмотр *ER*-диаграмм и повторное выполнение последующих этапов.

3.2. Логическое проектирование

Логической моделью БД называется модель логического уровня, построенная в рамках конкретной СУБД, в среде которой проектируется БД. Описание логической структуры БД в терминологии данной СУБД называется *схемой* БД. Под проектированием логической структуры БД понимается определение всех информационных единиц и связей между ними в терминологии выбранной СУБД, задание их имен, определение типов данных, задание по возможности количественных характеристик информационных единиц, таких как длины полей.

Таким образом, на этапе логического проектирования происходит преобразование концептуальной модели в модель, которая поддерживается выбранной СУБД. Очевидно, что такое преобразование не должно никаким образом сказаться на адекватности полученного результата исходной предметной области.

Для преобразования *ER*-модели в логическую существует строгая последовательность вполне определенных действий, что позволило во многих современных СУБД автоматизировать этот процесс в рамках CASE-технологий.

На первом шаге каждой сущности ставится в соответствие отношение (рис. 3.16). Атрибуты сущности при этом становятся атрибутами отношения, а ключевой атрибут (в случае, если он один) сущности становится первичным ключом отношения. При наличии же нескольких возможных ключей первичный ключ из них необходимо выбирать, руководствуясь следующими правилами.

Рис. 3.16. Алгоритм перехода к реляционной модели

При отсутствии каких-либо дополнительных ограничений в качестве первичного ключа выбирается возможный ключ (рис. 3.17), имеющий минимальный набор атрибутов (лучше всего, если это всего один атрибут). Однако зачастую в процессе функционирова-

ния БД значения ключевого атрибута могут меняться. В этом случае такой атрибут нежелательно выбирать в качестве первичного ключа, поскольку не все СУБД поддерживают возможность автоматического изменения связанных записей.

Иногда случаются ситуации, когда для задания первичного ключа отношения приходится использовать так называемые искусственные идентификаторы объекта-сущности. Делается это в следующих случаях.

Рис. 3.17. Алгоритм выбора первичного ключа

1. Наличие синонимов в предметной области. Это означает, что естественное свойство объекта, наиболее подходящее для использования его в качестве первичного ключа, не обладает свойством уникальности. Например, в списке сотрудников может быть по каким-то причинам не предусмотрено наличие таких свойств, как табельный номер или номер паспорта (эти поля, очевидно, были бы уникальными и могли бы использоваться в качестве ключей).

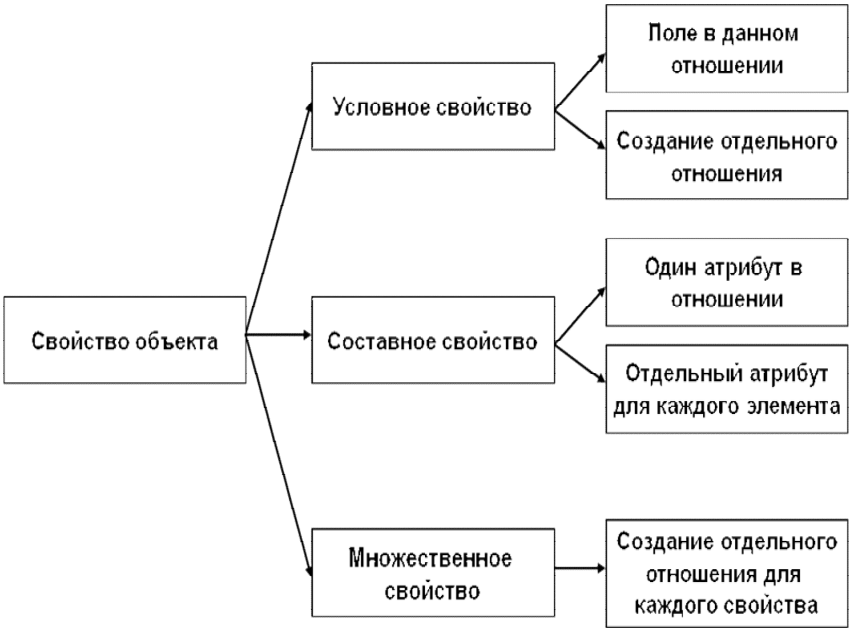


Рис. 3.18. Отображение свойств простых объектов

Естественный идентификатор сотрудника “ФИО” при наличии полных однофамильцев в этом случае теряет свойство уникальности и не может быть использован в качестве ключа. Единственным выходом из ситуации становится искусственное добавление дополнительного ключевого (например, автоинкрементного) поля.

2. Второй случай рассмотрен выше – значения естественного идентификатора объекта могут меняться со временем.

3. Один и тот же объект может участвовать во множестве связей с другими объектами. Это значит, что некий естественный идентификатор объекта должен присутствовать в качестве атрибута во многих отношениях. Если этот идентификатор является длинным (например, текстовым полем), а других более коротких естественных идентификаторов у этого объекта нет, то целесообразным является введение какого-то более короткого кодового обозначения для каждого экземпляра этого объекта.

Во всех рассмотренных случаях при введении искусственных идентификаторов нужно заранее предусмотреть, чтобы эти идентификаторы были статичными во времени.

Если у сущности имеются множественные свойства (рис. 3.18), то, как правило, для каждого такого свойства создается отдельное отношение, включающее два атрибута – атрибут, выбранный в качестве первичного ключа и атрибут, являющийся множественным свойством.

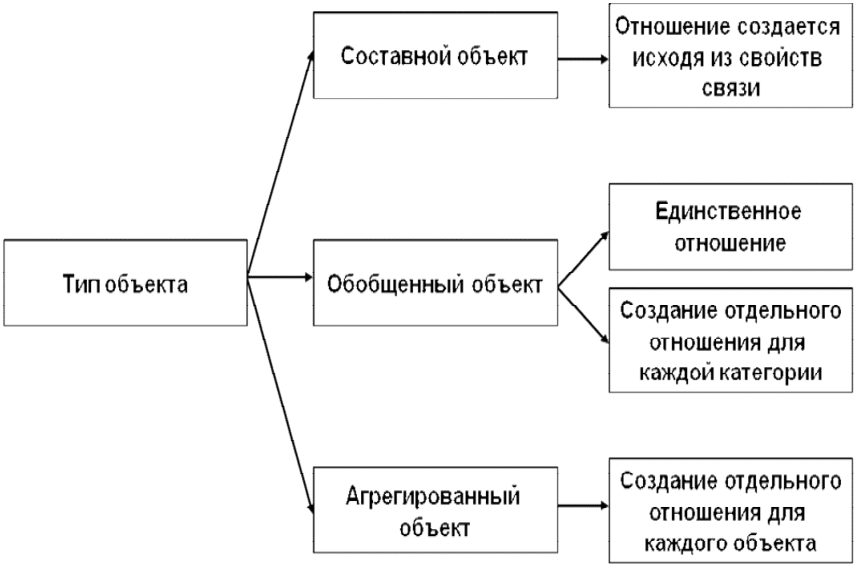


Рис. 3.19. Отображение свойств сложных объектов

При наличии условных свойств объекта (под объектом в данном случае подразумевается запись в отношении, то есть экземпляр сущности) оно может либо храниться таким же образом, как и обычные свойства объекта, т. е. в том же самом отношении, либо для этого свойства создается отдельное отношение, полями которого являются первичный ключ объекта и данное свойство. Первый случай оказывается предпочтительным, если этим свойством обладает множество объектов. Если же очень небольшое количество объектов имеют данное свойство, то множество значений будут

пустыми в соответствующем поле и предпочтительным оказывается второй вариант.

При наличии составных свойств имеется два варианта: либо всему свойству ставится в соответствие один атрибут отношения, либо каждому отдельному элементу свойства ставится в соответствие свой атрибут. Выбор того или иного варианта определяется исходя из дополнительных условий, вытекающих из особенностей обработки данных.

Отдельного анализа требует задача создания отношений в случае наличия в *ER*-модели сложных объектов (рис. 3.19).

Для составных объектов нужно в каждом случае исходить из степени связи между ними и класса ее принадлежности, после чего создавать отношения в соответствии с рассмотренными ранее правилами.

Для обобщенных объектов также не существует однозначного решения. Можно всему объекту поставить в соответствие одно отношение, в котором будут перечислены атрибуты всех категорий этого объекта. В таком отношении обязательно должен присутствовать атрибут, по которому происходит деление объекта на категории.

Другим вариантом является создание для каждой категории объекта собственного отношения с атрибутами, соответствующими наследуемым и уникальным свойствам данной категории.

При наличии агрегированных объектов каждому такому объекту ставится в соответствие определенное отношение. Атрибутами этого отношения становятся ключевые свойства всех входящих в агрегированный объект сущностей, и все остальные свойства этих сущностей.

Для каждого атрибута отношения устанавливается тип данных, указывается свойство уникальности, обязательность или необязательность задания значения (атрибуты первичного ключа всегда должны иметь свойство обязательности) и при необходимости значения по умолчанию (рис. 3.20). На этом этапе происходит переименование имен сущностей и их атрибутов в соответствии с синтаксическими особенностями выбранной СУБД.

Связи между сущностями переносятся на связи между реляционными отношениями по правилам, изложенным в предыдущем

разделе. При добавлении ключевых атрибутов одного отношения в набор атрибутов другого эти атрибуты становятся в нем внешним ключом, причем в случае необязательного класса принадлежности связи у атрибутов внешнего ключа допустимо свойство необязательности значений, а при обязательности связи значения атрибутов внешнего ключа должны всегда быть определены.

Рис. 3.20. Определение свойств атрибутов отношения

Итак, результатом логического проектирования БД является разработка ее логической схемы на языке описания данных выбранной СУБД. Возникает вопрос, как оценить полученный результат, какие критерии нужно использовать для того, чтобы сделать вывод о том, что разработанная схема БД является удовлетворительной? В одних случаях, например, на первом месте находится скорость доступа к данным, в других – степень надежности защиты информации, в третьих – простота обслуживания и т. д. При этом во многих случаях один критерий может входить в прямое противоречие с другим: увеличение скорости доступа к данным зачастую означает понижение степени защиты данных и ставит под угрозу безопасность информационной системы.

Ясно, что все критерии должны быть взаимосвязаны друг с другом, а оценка разработанной БД должна быть комплексной по совокупности всех критериев. При этом, конечно же, значимость каждого из критериев в зависимости от контекста использования информационной системы может быть разной, т. е. каждый критерий имеет свой вес.

В настоящее время имеется стандартный набор основных критериев, которые используются при анализе БД.

1. Соответствие разработанной БД исходной предметной области. Этот критерий называется адекватностью схемы БД. Данное требование является исходным, поскольку без его выполнения говорить о чем-либо другом просто бессмысленно. Любое искажение предметной области означает, что БД является некорректной и не может использоваться.

2. Одним из проявлений адекватности БД является ее полнота – возможность удовлетворения потребностей различных категорий пользователей не только на данный момент времени, но и в дальнейшем. Под этим понимается также и то, что потребности пользователей могут быть самыми разнообразными и не всегда на стадии проектирования БД можно их все заранее спрогнозировать. Поэтому БД может хранить в себе большое количество детализированных описаний объектов и связей между ними.

3. Сложность структуры БД. Это требование относится прежде всего к структурированным БД, и в первую очередь к реляционным моделям. Под степенью сложности реляционной модели понимается количество отношений в БД, число атрибутов в каждом отношении, количество различных индексов, ключевых атрибутов, характер связей между отношениями. Как правило, БД считается тем лучше, чем проще ее схема. Однако это далеко не всегда так и зачастую излишнее упрощение схемы БД может привести к нарушению корректности ее схемы (за примером можно обратиться, например, к необходимости нормализации отношений).

4. Адаптируемость. Это требование трактуется достаточно широко. Под адаптируемостью понимается прежде всего возможность БД воспринимать изменения, происходящие в предметной области. Если в предметной области происходят изменения, то схема БД должна быть такой, чтобы эти изменения не отражались на струк-

туре и логической модели БД. Если же по каким-то причинам обойтись без изменения структуры БД все же невозможно, то критерием оценки схемы БД является простота и эффективность внесения этих изменений.

Кроме того, с адаптируемостью связана способность системы учитывать возможность изменения потребностей различных категорий пользователей. С этим связана прежде всего возможность удовлетворять нерегламентированные запросы пользователей. Для этого в БД должны храниться детализированные данные, позволяющие получать различный набор производных данных в соответствии с разнообразными критериями запросов. Это означает, что пользователи могут получать информацию, отличную от непосредственно хранимой в БД.

Еще одним аспектом является возможность адаптации системы к изменениям в программном обеспечении. Это означает, что программное обеспечение должно быть стандартизированным и широко распространенным.

5. Дублирование данных. Дублирование данных является необходимым для обеспечения целостности данных в случае возникновения потребности в восстановлении БД. Однако необходимо понимать, что дублирование приводит к увеличению объема памяти, к затратам на поддержание идентичности всех имеющихся копий, к требованиям, накладываемым на программное обеспечение, а значит, и к усложнению всей системы.

6. Объем необходимой памяти.

7. Скорость доступа к данным и обработки информации. Этот критерий относится прежде всего к системам, работающим в реальном масштабе времени. Оценить его иногда бывает достаточно сложно, особенно если система работает в многопользовательском режиме. Легче всего это сделать с помощью специальных тестовых программ, учитывающих большое количество как взаимозависимых, так и взаимонезависимых факторов.

8. Универсальность. Под этим, в частности, понимается возможность использования БД широким кругом пользователей, в том числе не являющихся специалистами в области компьютерных технологий.

3.3. Физические модели баз данных

Под физической моделью БД понимают способ размещения данных на устройствах внешней памяти и способ доступа к этим данным. Каждая СУБД по-разному организует методы хранения и доступа к данным, однако поскольку в подавляющем большинстве случаев СУБД работает под управлением конкретной операционной системы, эти методы в большой степени зависят от методов работы с данными самой операционной системы.

В настоящее время имеются два основных способа хранения данных. Первый основан на *файловых системах*, а второй – на *страничных моделях* организации данных (рис. 3.21).

Рис. 3.21. Способы хранения данных

Способ организации файловой структуры проиллюстрирован на рис. 3.22. Файл представляет собой последовательность записей постоянной либо переменной длины. В первом случае для разме-

щения каждой записи выделяется фиксированный объем памяти, который оговаривается заранее. К таким записям относятся, в частности, некоторые числовые и символьные типы данных. В случае записей переменной длины каждая запись может быть произвольного объема. К таким записям можно отнести текстовые типы данных. В том и другом случаях доступ к нужной записи является последовательным, так как для начала считывания данной записи необходимо считать длины всех предыдущих записей.

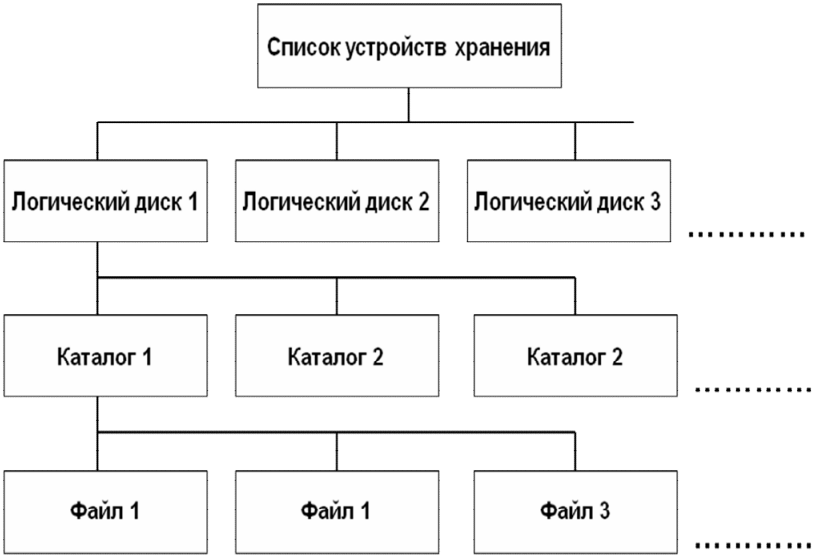


Рис. 3.22. Организация файловой структуры

В соответствии с типом организации файла различают стратегии последовательного и произвольного размещения записей (адресации записей). При последовательном размещении записей каждая следующая запись физически располагается непосредственно за предыдущей. При произвольной адресации адрес начала каждой записи определяется исходя из определенных факторов, к которым прежде всего относятся факторы скорости считывания записи.

Среди способов организации файловых структур рассмотрим те методы, которые предназначены для оптимизации доступа к данным, а следовательно, для увеличения производительности всей

системы. Под доступом к файлом будем понимать способ перемещения записей из внешней памяти в оперативную.

При страничной организации доступа данные перемещаются страницами фиксированной длины. При этом размер страницы определяется СУБД и в ее задачу входит такая организация записей, при которой эффективность заполнения страниц оказывается наилучшей. В случае параллельной секционной организации доступа к данным в системе должно быть несколько независимых механизмов доступа, которые могут работать в параллельном режиме.

Поиск записи в файле может осуществляться по ее номеру, однозначно идентифицирующего адрес записи (рис. 3.23). Файлы с фиксированной длиной записи обеспечивают наибольшую скорость доступа, поскольку в этом случае адрес каждой записи может быть непосредственно вычислен исходя из номера записи и длины записи.

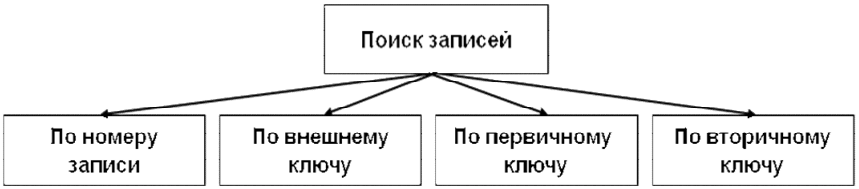


Рис. 3.23. Способы поиска записей в файле

Очень часто поиск информации проводится по значениям первичного или внешнего ключей, когда номер соответствующей записи заранее неизвестен. Ключом является некое значение, которое располагается в каждой записи в одной и той же позиции. Иногда (в очень редких случаях) можно построить некую функцию, которая устанавливает взаимно-однозначное соответствие между значением ключа и номера записи.

Однако, как правило, такую функцию построить принципиально невозможно. В таких случаях используется так называемый метод хэширования. Суть этого метода заключается в том, что по значению ключа вычисляется хэш-функция, значение которой берется для начала поиска. При этом взаимно-однозначного соответствия между значением хэш-функции и номером записи не устанавливается – одно и то же значение хэш-функции соответствует нескольким записям. Такая ситуация называется коллизией, а запи-

си, имеющие одно и то же значение хэш-функции, называются синонимами. Таким образом, при использовании метода хэширования решаются две задачи.

1. Выбор хэш-функции. Признаком “хорошей” хэш-функции является возвращение ею не более 10 синонимов.

2. Выбор стратегии разрешений коллизий.

Далеко не всегда удается найти подходящую хэш-функцию. Поэтому в таких случаях предпочтительным является использование индексных файлов. Такие файлы состоят из двух частей – индексной и основной. Физически, как правило, эти части реализуются в отдельных файлах. Индексная часть строится для основной части, в которой размещены непосредственно сами записи. Различаются *файлы с плотным индексом* (*индексно-прямые файлы*), *файлы с неплотным индексом* (*индексно-последовательные файлы*) и *B-деревья*. Все типы перечисленных индексных файлов предназначены для поиска записей по значению первичного ключа.

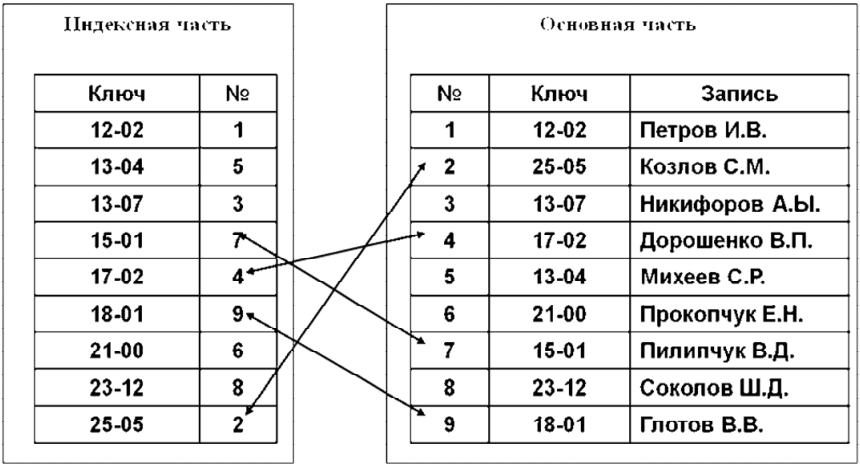


Рис. 3.24. Структура файла с плотным индексом

В файлах с плотным индексом в основной части хранятся записи фиксированной длины (рис. 3.24). В индексной части каждая запись содержит значение первичного ключа и порядковый номер записи из основной части, имеющей данное значение первичного ключа. Таким образом, в файлах с плотным индексом каждой запи-

си в основной части соответствует единственная запись в индексной, и наоборот. Все записи в индексной части упорядочены по значению ключа, а в основной части расположены в произвольном порядке. Поиск записи проводится в индексной части. Этот поиск является двоичным (бинарным), при котором сначала находится запись, расположенная в середине индексной части, и значение ее ключа сравнивается со значением, введенным в качестве поискового. Далее определяется та половина индексной части, в которой находится искомое значение ключа, находится запись, расположенная в ее середине, и процесс повторяется до нахождения нужной записи. После этого по порядковому номеру записи основной части, соответствующей найденному значению ключа, осуществляется доступ к искомой записи основной части.

Рис. 3.25. Структура файла с неплотным индексом

При добавлении новой записи она записывается в конец основной части. В индексной части для сохранения ее упорядоченности происходит модификация записей путем вставки новой записи в нужное место. При удалении записи в индексной части запись уничтожается физически со сдвигом всех нижерасположенных записей на одну позицию. В основной части запись физически не уничтожается, а помечается как несуществующая.

В индексно-последовательных файлах, наоборот, записи в основной части упорядочены (рис. 3.25). Объем внешней памяти, где хранится основная часть файла, разделен на блоки, в каждом из которых находится определенное количество записей. Запись в индексной части хранит информацию о значении первичного ключа записи, расположенной первой в данном блоке, и номер этого блока. Это означает, что в индексной части по значению первичного ключа проводится двоичный поиск нужного блока. После нахождения искомого блока осуществляется двоичный поиск среди записей, находящихся в нем.

При добавлении новой записи она заносится в основную часть на нужное место. При удалении запись физически уничтожается в основной части файла, а индексная часть при этом, как правило, не модифицируется.

Основным достоинством использования файлов с неплотным индексом является увеличение скорости доступа к данным по сравнению с использованием файлов с плотным индексом. Однако необходимо учитывать, что организация файлов с последовательным индексом требует определенных затрат на поддержание упорядоченности записей в основной части.

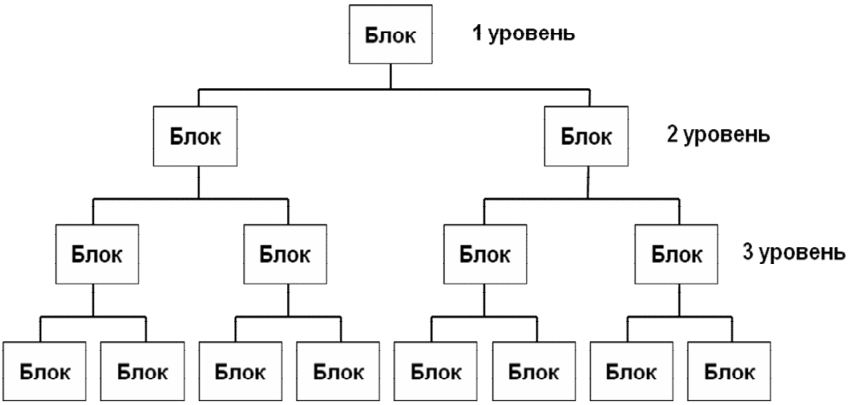


Рис. 3.26. Структура B-деревьев

Файлы в виде B-деревьев (рис. 3.26) основаны на построении неплотного индекса над уже имеющимся неплотным индексом, т. е. индексная часть файла становится основной частью для индексной

части более высокого уровня. Таким образом, получающаяся структура имеет вид сбалансированного графа (дерева), в котором каждый блок одного уровня связан с одним и тем же количеством блоков более низкого уровня и с одним блоком более высокого уровня.

Очевидно, что порядок поиска, удаления и добавления записи в этом случае такой же, как и для файлов с неплотным индексом, скорость доступа к данным еще более возрастает, а основные трудности при организации такой файловой структуры связаны с необходимостью упорядочивания записей на каждом уровне.

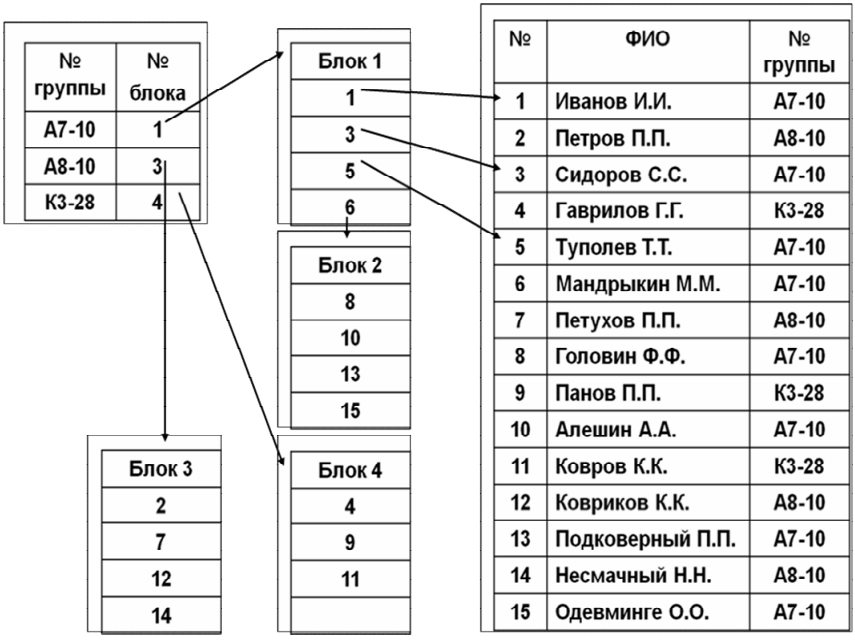


Рис. 3.27. Структура инвертированного списка

В отличие от рассмотренных индексных файлов, записи в которых ищутся по значениям первичных ключей, файлы, называемые инвертированными списками, предназначены для поиска данных по значениям вторичных ключей. Это означает, что в результате поиска находится некоторое количество записей, имеющих одно и то же значение вторичного ключа.

Инвертированный список состоит из трех частей (рис. 3.27). В основной части находятся сами записи. Индексная часть первого уровня состоит из блоков, в каждом из которых содержатся номера записей основной части, имеющих одно и то же значение вторичного ключа. Все блоки упорядочены по значениям вторичных ключей. В индексной части второго уровня каждая запись содержит значение вторичного ключа и соответствующий ему номер блока. Эта часть также упорядочена по значениям вторичных ключей.

Поиск нужных записей осуществляется следующим образом. Сначала по номеру вторичного ключа находятся блоки, в которых содержатся номера искомых записей. После этого происходит считывание этих записей.

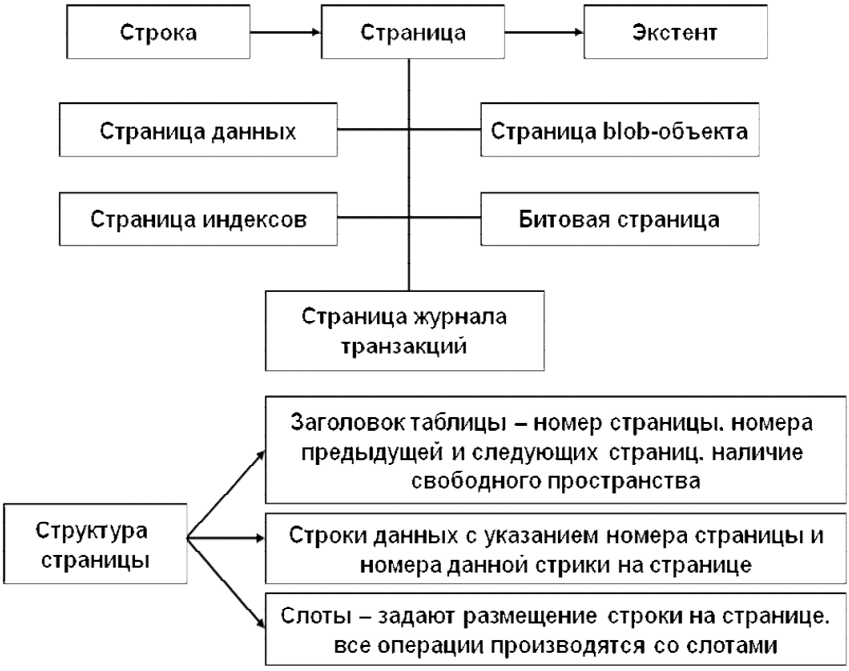


Рис. 3.28. Страничная организация данных

Использование инвертированных списков в большинстве случаев ускоряет поиск данных по значениям вторичных ключей. Од-

нако иногда наличие большого количества инвертированных списков в одной информационной системе при условии, что обновление вторичных ключей происходит в ней достаточно часто, может привести к замедлению работы системы в целом.

Таким образом, при файловой организации данных БД состоит из множества различных файлов – основных и индексных. Кроме того, для различных целей могут использоваться такие файловые структуры, как файлы метаданных, файлы-указатели и др. Поскольку именно с такими структурами работает операционная система, то СУБД вынуждена подчиняться принципам работы, заложенным в данную операционную систему. С точки зрения достижения максимальной эффективности обслуживания БД эти принципы не всегда являются оптимальными. Нужно, чтобы при обслуживании БД система управления забирала всю внешнюю память, на которой расположена БД, а операционная система не имела доступа к этой части дискового пространства.

Все это привело к бесфайловым страничным структурам организации данных (рис. 3.28). В такой структуре физическое распределение дискового пространства происходит следующим образом. Основной единицей хранения информации является страница данных, данные на которой записываются в виде строк. Совокупность нескольких страниц фиксированной длины образуют экстент. Каждый новый экстент создается после заполнения предыдущего и связывается с ним указателем. Минимальный размер экстента, как правило, равен четырем страницам.

Каждая страница имеет вполне определенную структуру, которая включает в себя следующие элементы.

1. Заголовок страницы содержит в себе номер данной страницы, номера предыдущей и следующей страниц, а также информацию о наличии свободного пространства.

2. Содержание страницы представляет собой непосредственно сами записи данных в виде строк, каждая из которых имеет свой номер.

3. Дескрипторы данных (слоты) определяют длину строки и размещение строки на странице. Это необходимо для того, чтобы при необходимости переупорядочения строк на странице не делать это физически – все манипуляции происходят только со слотами.

Кроме того, имеются специальные страницы индексов и страницы blob – объектов, предназначенные для хранения слабоструктурированных данных, таких как тексты большого объема, графические и видеоизображения и др. Имеются также битовые страницы, используемые для трассировки других страниц, и страницы журнала транзакций.

Что касается физического размещения данных во внешней памяти, то практически всегда для этого используется несколько дисков. Так, на одном диске может располагаться операционная система, на втором – сами данные, на третьем и четвертом – журнал транзакций и индексная система соответственно.

3.4. CASE-технологии

В последнее время широкое распространение получила технология автоматизированного проектирования информационных систем (ИС) и программных продуктов, называемая CASE-технологией (Computer Aided System Engirneering). Понятие CASE включает в себя совокупность регламентно-методических материалов, автоматизированных методов и инструментальных средств разработки, поддерживающих все этапы Жизненного Цикла Системы (ЖЦС, Bysiness System Life Cycle) начиная от первоначального формирования технических требований и спецификаций до получения и сопровождения готового программного продукта. Таким образом, понятие CASE охватывает всевозможные методические и программно-инструментальные средства, объединенные общей целью эффективной поддержки любых процессов создания прикладного программного обеспечения. Использование CASE-технологии при проектировании сложных ИС обеспечивает существенное увеличение производительности труда на всех этапах разработки и значительно сокращает затраты на сопровождение и модификацию по сравнению с проектированием ИС вручную.

В состав любой CASE-системы входят: метод, нотация и средство (рис. 3.29). Под методом подразумевается методология проектирования. Нотацией называется выбранное для описания системы средство, которым является программный продукт либо компьютерный инструмент, предназначенный для поддержки и усиления

методов проектирования (т. е. та среда, в которой работает пользователь).

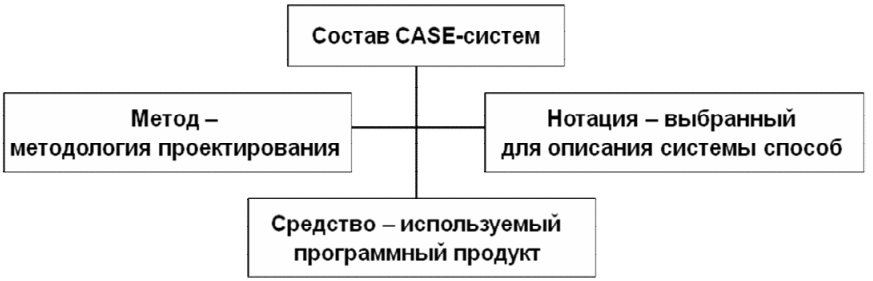


Рис. 3.29. Состав CASE-систем

Основными функциональными компонентами CASE-систем являются следующие (рис. 3.30):

- централизованное хранилище информации о проекте (БД проекта);
- средства ввода данных в БД;
- средства анализа и редактирования информации в БД;
- средства вывода (средства документирования и верификации).

CASE-средства обеспечивают необходимый для проектирования ИС набор средств. Эти средства включают технологии и стандарты построения диаграмм БД, методы автоматического определения ошибок, а также комплекс технологий для конструирования программных систем.

Рис. 3.30. Компоненты CASE-систем

CASE-средства позволяют решать ряд задач в рамках ЖЦС, к которым относятся задачи стратегического планирования проекта, моделирования ПО, изучения возможных вариантов решения проблем, определения требований к ИС, системного проектирования, программирования, тестирования, отладки программного обеспечения и измерения качества, поддержки документирования, управления процессом проектирования, сопровождения (рис. 3.31).

Рис. 3.31. Задачи CASE-систем

Современные CASE-системы имеют в своем арсенале центральный словарь и БД проекта, средства генерации кода, средства управления проектом, словарь проекта, средства генерации отчетов и другие средства, используемые на различных этапах создания или модификации системы.

Наиболее общими характеристиками CASE-систем являются следующие (рис. 3.32): поддержка единой БД проекта, поддержка одновременной работы группы разработчиков, поддержка полного ЖЦС, поддержка визуальных методов проектирования, автоматизация кодирования, информационное обеспечение разработчиков, документирование проекта, управление проектом, возможности тестирования и отладки, возможности повторной разработки системы и интеграции различных систем, открытая архитектура.

Рис. 3.32. Характеристики CASE-систем

К сожалению, ни одна из существующих в настоящее время CASE-систем не имеет всех возможностей из этого списка.

В зависимости от вида проектных описаний, специфицируемых в методологиях CASE-систем, они делятся на структурные, функциональные и смешанные, в том числе объектно-ориентированные

(рис. 3.33). *Структурные методологии* определяют структуру информационного фонда вне зависимости от процессов его обработки. В *функциональных методологиях* первичными являются процессы (функции), которые последовательно детализируются до уровня программных модулей. В *смешанных методологиях* предполагается параллельное взаимосвязанное проектирование структур данных и процессов их обработки.

Рис. 3.33. Виды CASE-систем

Методологии классифицируют также по их принадлежности различным этапам ЖЦС. В соответствии с этим они делятся на верхние, средние и нижние (рис. 3.34).

Верхние CASE, иногда называемые системами компьютерного планирования, позволяют строить модель ПО, отражающую меха-

низмы функционирования, имеющиеся возможности, ресурсы, цели проекта.

Средние CASE – это средства детального описания структуры создаваемого программного обеспечения. Они позволяют анализировать информационные и конструктивные проблемы, используют диаграммы и словари данных, существенно сокращая время разработки проекта. Немаловажную роль при этом играет возможность накопления и хранения полученной информации, что позволяет использовать эти решения при создании других проектов.

Рис. 3.34. Методологии CASE-систем

Нижние CASE – это средства, поддерживающие системы разработки программного обеспечения. Они используют компоненты генерации программ, создания документации и тестирования.

Любая методология CASE-технологии базируется на концепции ЖЦС. Согласно этой концепции проектируемая система проходит в своем развитии ряд вполне определенных этапов, каждый из которых ставит свои задачи перед разработчиком. Каждый этап является внутренне завершенным и служит основой для последующих этапов. На каждом этапе применяются определенные методы и инструментальные средства.

Рис. 3.35. Этапы жизненного цикла систем

В настоящее время, несмотря на имеющиеся различия существующих методологий, эти этапы практически идентичны для всех моделей ЖЦС (рис. 3.35): Стратегия, Анализ, Проектирование, Реализация, Документирование, Внедрение. В совокупности они и составляют ЖЦС, который отражает стадии развития, необходимые для создания системы любой сложности. В настоящее время наиболее широко распространенными являются каскадная, спи-

ральная модели ЖЦС и модель с промежуточным контролем (рис. 3.36).

Рис. 3.36. Модели жизненного цикла систем

В *каскадной модели* предполагается строгое последовательное выполнение перечисленных этапов. Несмотря на то, что при таком подходе обычно проще осуществлять контроль над процессом проектирования ИС, регулировать затраты и планировать мероприятия, сам процесс создания ИС является итерационным: после выполнения определенного этапа происходит пересмотр принятых решений и возврат к предыдущим этапам проектирования. Это является одним из основных недостатков каскадной модели.

Для того, чтобы максимально согласовать ЖЦС с реальным процессом проектирования ИС, используется *модель с промежуточным контролем*, в которой допускается возврат к более ранним этапам. Однако в этом случае недостатком становится увеличение длительности процесса проектирования.

В *спиральной модели* оказывается возможным начало работ на следующем этапе даже при неполном завершении предыдущего этапа. Тем самым эта модель устраняет основные недостатки двух других.

На первом этапе ЖЦС определяется стратегия разработки ИС. Для этого в процессе диалога с заказчиком формулируются наиболее общие требования к будущей системе, производится постановка задачи, выбирается наилучший вариант “архитектуры системы”, т.е. выделяются наиболее важные направления, на которые нужно обратить внимание при ее последующей разработке. Результатом этого этапа должен быть комплекс моделей, четко описывающих задачи и информационные потребности организации в целом, комплекс рекомендаций и план разработки ИС, которая должна удовлетворять как текущие, так, по возможности, и будущие информационные потребности организации с учетом организационных, финансовых и технических ограничений.

Цель этапа анализа – показать, как реально работает организация и как достигаются ее цели. В большинстве систем на этом этапе применяется метод структурного анализа. Структурный анализ является развитием системного анализа применительно к системам обработки информации.

Данный метод строится на наглядной диаграммной технике – для описания модели проектируемой системы используются диаграммы, схемы и структурограммы.

Чаще всего в структурном анализе применяются следующие диаграммы:

- диаграммы потоков данных – описание движения информационных потоков между различными подсистемами, накопителями информации, внешними источниками и потребителями информации;
- *ER*-диаграммы – выявление основных объектов ПО, отношений между ними и их свойств;
- диаграммы переходов состояний – одна или несколько связанных диаграмм дают разработчикам достаточно полное представление о структуре ПО и о происходящих там процессах.

Таким образом, результатом выполнения первых двух этапов является построение концептуальной модели ПО.

На этапе проектирования концептуальная модель, разработанная на предыдущих этапах, преобразуется в логическую модель, поддерживаемую конкретной СУБД.

На стадии реализации создается и тестируется реальная система.

Завершается цикл этапом внедрения и производства, на котором производится тестирование системы разработчиками и заказчиками и запуск ее в эксплуатацию.

С момента своего появления CASE-средства прошли большой путь развития. Первый этап эволюции этих средств совпал с развитием средств программирования. Сконцентрировавшись на принципах проектирования, разработчики изыскивали пути создания более экономного и надежного программного обеспечения. Средства (компиляторы и отладчики) и методы (структурный анализ и дизайн) помогали упростить разработку, увеличивая точность и скорость.

Следующим этапом было появление инструментальных средств программного проектирования. Средства внешнего интерфейса позволили использовать компьютерную графику для представления объектов программирования. Внутренние средства используют возможности компьютера для выделения детализированной информации из более общей.

В результате CASE-средства позволили автоматизировать разработку полного ЖЦС. Такие средства получили название “Интегрированный CASE” (Integrated CASE). Термин “интегрированный” означает, что различные CASE-средства, поддерживающие ЖЦС, работают согласованно, как если бы они были компонентами одной и той же среды.

Современные CASE-средства реализуют следующие группы функций (рис. 3.37):

- ведение всех видов проектных описаний для различных этапов разработки в режиме их “ручного” определения (через графические диаграммы и/или экранные формы);
- верификация проектных спецификаций;
- документирование;
- автоматическая генерация некоторых видов проектных описаний;

- спецификация интерфейса с будущими пользователями ИС в виде создания экранных форматов планируемого представления информации;
- возможность параллельной работы проектировщиков над общим проектом;
- разграничение доступа проектировщиков;
- простой и удобный интерфейс.

Первоначально опыт поддержки этих функций был получен на больших ЭВМ. Только последние достижения персональной компьютерной техники создали среду для эффективной реализации удобных в эксплуатации CASE-средств. В эту среду вошли персональные СУБД с языками 4-го поколения (4GL), графический и текстовой редакторы, режимы сетевой работы и новые оконные интерфейсы.

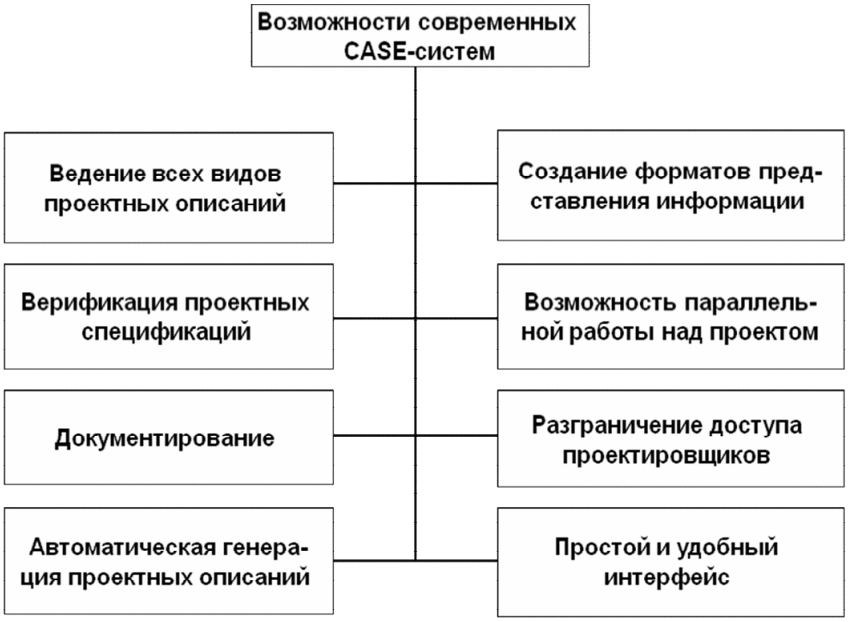


Рис. 3.37. Возможности CASE-систем

В заключение данного раздела перечислим основные используемые в настоящее время CASE-технологии.

1. Системы Design/IDEF и BPwin используются как средства анализа и предназначены для исследования всевозможных моделей предметной области.

2. Основными средствами проектирования баз данных являются такие технологии, как ERwin, S-Designor, DataBase Designor.

3. Технологии PRO-IV, Vantage Team Builder применяются для решения задач создания различных проектных спецификаций в качестве средств анализа и проектирования.

4. Технологии Oracle, JAM, Delphi, C++ Duilder, Uniface, Power Builder, SQL являются основными средствами для разработки приложений. Эти средства могут быть предназначены как для решения задач на одном или нескольких этапах ЖЦС (такие как Erwin, S-Designor), так и являться интегрированными, поддерживающими весь ЖЦС (Vantage Team Builder, Oracle).

Среди перечисленных CASE-технологий можно выделить структурные CASE-системы (такие, как Vantage Team Builder), основанные на используемых в них методах структурного анализа и программирования, объектно-ориентированные (Object Team) и комбинированные CASE-системы (Oracle).

Контрольные вопросы

1. На каком этапе проектирования базы данных разрабатывается ее концептуальная модель? Что она из себя представляет и каковы ее основные свойства?

2. Что такое ER-модель?

3. Дайте определение сущности. Что такое атрибут сущности и экземпляр сущности?

4. Какими бывают связи между сущностями с точки зрения степени связи и класса принадлежности связи? Приведите примеры

5. Какие атрибуты сущности называются ключевыми?

6. Что такое диаграммы *ER*-экземпляров и *ER*-диаграммы?

7. Что называется классом объектов?

8. Что называется идентификатором объекта?

9. Какие свойства объекта называются единичными, а какие – множественными?

10. В чем заключается различие между реальными и абстрактными объектами? Приведите примеры.

11. Какие свойства объекта называются статическими, а какие – динамическими? Приведите примеры.

12. В чем заключается различие между простыми и сложными объектами?

13. Приведите примеры составных, обобщенных и агрегированных объектов.

14. В каких случаях при переходе от модели “сущность-связь” к реляционной модели двум взаимосвязанным сущностям ставится в соответствие одно, два, три отношения?

15. Перечислите основные этапы проектирования базы данных в рамках концептуального моделирования.

16. Что называется логическим проектированием базы данных?

17. Перечислите этапы логического проектирования?

18. Что называется искусственным идентификатором сущности? Для каких целей и в каких случаях он используется?

19. Перечислите основные критерии, которые используются при анализе базы данных на этапе логического проектирования?

20. Что понимается под физическим моделированием базы данных?

21. Какие существуют способы хранения данных?

22. В чем заключаются стратегии последовательного и произвольного размещения записей?

23. Что следует понимать под скоростью доступа к данным?

24. В чем заключается сущность метода хэширования? Что такое хэш-функция?

25. Для каких целей используются индексные файлы?

26. Что общего между индексно-прямыми и индексно-последовательными файлами? В чем заключаются принципиальные отличия?

27. Каким образом реализуется механизм добавления и удаления записей в индексно-прямых и индексно-последовательных файлах?

28. Какие файлы называются B-деревьями?

29. Для каких целей используются инвертированные списки? Приведите примеры.

30. В чем заключаются особенности страничных бесфайловых структур организации данных?

31. Что понимается под CASE-технологиями и CASE-средствами?

32. Какие компоненты включают в себя CASE-системы?

33. Какими общими характеристиками обладают современные CASE-системы?

34. Сформулируйте свойства, области применения верхних, средних и нижних CASE-системы.

35. Что называется жизненным циклом системы? Из каких этапов оно состоит?

36. Какие существуют модели ЖЦС? Сравните их между собой

37. Перечислите основные используемые в настоящее время CASE-технологии. Для решения каких задач используется та либо иная CASE-технология.

4. УПРАВЛЕНИЕ БАЗАМИ ДАННЫХ, СТАНДАРТ SQL

4.1. Целостность баз данных

Понятие *целостности* является одним из основополагающих в теории БД. Любая БД содержит в себе информацию об объектах реального мира, находящихся в связи между собой. Под целостностью БД понимается соответствие модели предметной области, хранимой в БД, объектам реального мира и их связям между собой в каждый момент времени. Целостность бывает физической и логической (рис. 4.1).

Рис. 4.1. Виды целостности баз данных

Под *физической целостностью* понимается возможность физического доступа к данным в любой момент времени, а также то, что эти данные не утрачены. *Логическая ценность* означает отсутствие логических ошибок в БД – нарушение структуры всей БД или отдельных объектов, удаление или некорректное изменение связей между объектами и т. д.

Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий. В реляционной модели имеется три основных критерия, обеспечивающие поддержку логической целостности БД.

1. Поддержка *структурной целостности*. Это означает, что любая реляционная БД строится только над структурами данных, удовлетворяющих свойствам отношений.

2. Поддержка *языковой целостности*. Любая реляционная СУБД должна иметь возможность описывать данные и манипулировать ими в формате не ниже стандарта SQL. Это означает, что доступ к информации БД может быть выполнен только при помощи операторов SQL.

3. Поддержка *ссылочной целостности*. Этот принцип гласит, что при изменении данных во взаимосвязанных отношениях может произойти только одно из следующих действий:

- кортежи подчиненного отношения должны уничтожаться при удалении кортежа основного отношения, связанного с ними;
- кортежи основного отношения можно удалять только при отсутствии связанных с ними кортежей подчиненного отношения;
- кортежи подчиненного отношения при удалении кортежа основного отношения, связанного с ними, не удаляются, но модифицируются таким образом, что на месте ключа родительского отношения устанавливается значение NULL.

Ссылочная целостность поддерживает БД в непротиворечивом состоянии при модификации данных в процессе их добавления или удаления.

Кроме рассмотренных критериев, вводится также понятие *семантической целостности* БД. Обеспечение семантической целостности модели БД является одной из наиболее важных задач. Этот механизм называется механизмом ограничения целостности. Ограничения целостности означают по своей сути возможность или, наоборот, невозможность отдельным информационным единицам, содержащимся в БД, принимать те или иные значения и участвовать в связях с другими информационными единицами.

Ограничения семантической целостности могут задаваться либо декларативным (при описании БД), либо процедурным (в програм-

мах обработки данных) путями. Имеются следующие виды декларативных ограничений целостности (рис. 4.2).

1. Ограничения *целостности атрибута* – устанавливаются ограничения на значение по умолчанию, уникальность значения, на тип и формат значений, диапазон значений, на указание признака определенного или неопределенного значения, на домен, соответствующий данному атрибуту.

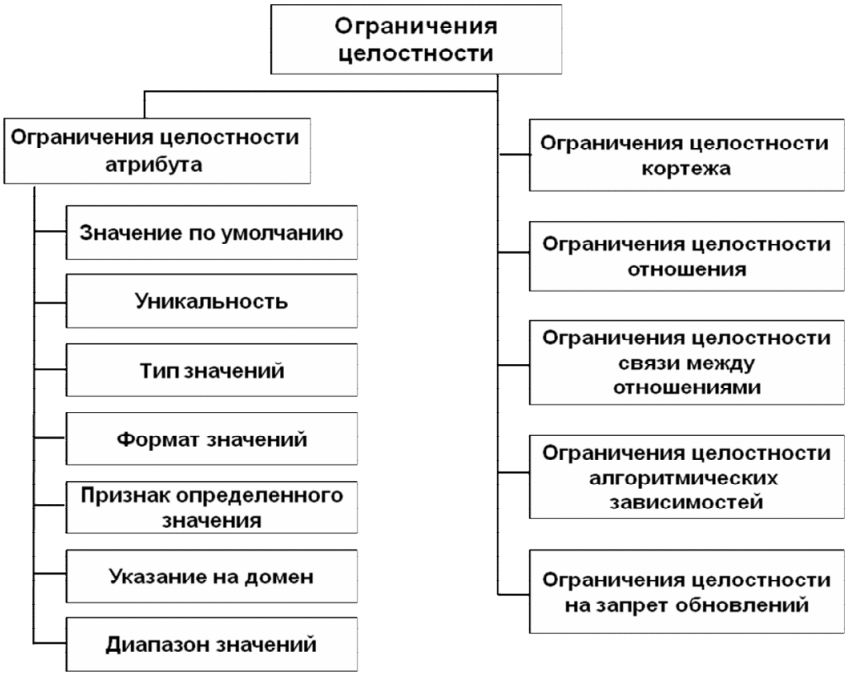


Рис. 4.2. Виды декларативных ограничений целостности

При задании значения по умолчанию каждый раз при вводе новой записи в таблицу при отсутствии прямого указания на значение атрибута этому атрибуту присваивается заданное значение.

Ограничение на уникальность обычно устанавливается тогда, когда уникальное значение является идентификатором некоторого объекта. Поэтому признак уникальности часто соответствует первичному ключу отношения (в случае, если ключ не является составным). При установлении ограничения на уникальность прове-

ряется допустимость данного значения путем просмотра всей таблицы.

Тип значения определяет допустимые для данного атрибута символы (числа, буквы, логические переменные и т. д.), а формат устанавливает более жесткие ограничения на возможные значения (например, формат “дата”).

Ограничение диапазона значений устанавливается в большинстве случаев для числовых типов данных. Диапазоны бывают односторонние (указывается только нижняя или верхняя границы) или двусторонние. Односторонний диапазон всегда является открытым, а двусторонний диапазон может быть как открытым, так и закрытым.

Признак определенного значения (обязательность заполнения) не допускает пустого значения атрибута.

Ограничение на домен означает, что атрибут может принимать значения только из определенного множества значений.

2. Ограничения *целостности кортежа*. Под этим подразумеваются возможные ограничения на соотношения значений отдельных атрибутов в пределах одной строки таблицы. Например, дата возврата книгу в библиотеку не может предшествовать дате взятия ее читателем.

3. Ограничения *целостности отношения*. В этом случае устанавливаются ограничения на соотношения данных, находящихся в разных строках таблицы. Примером может служить проверка на уникальность первичного ключа.

4. Ограничения *целостности взаимосвязанных отношений* – ограничения целостности связи и ограничения по существованию.

Ограничение *целостности связи* означает, что значение внешних ключей таблицы, отражающей связь между двумя объектами, всегда должны соответствовать одному из значений первичных ключей таблиц, описывающих эти объекты.

Под ограничением *целостности по существованию* понимается, что для существования данного объекта в данной таблице необходимо, чтобы он был связан с определенным объектом в другой таблице. Это ограничение является более сильным по сравнению с ограничением по связи, поскольку в этом случае устанавливается обязательный класс принадлежности связи. Удаление записей под-

чиненной таблицы никогда не должно приводить к нарушению ограничения целостности по связи или по существованию.

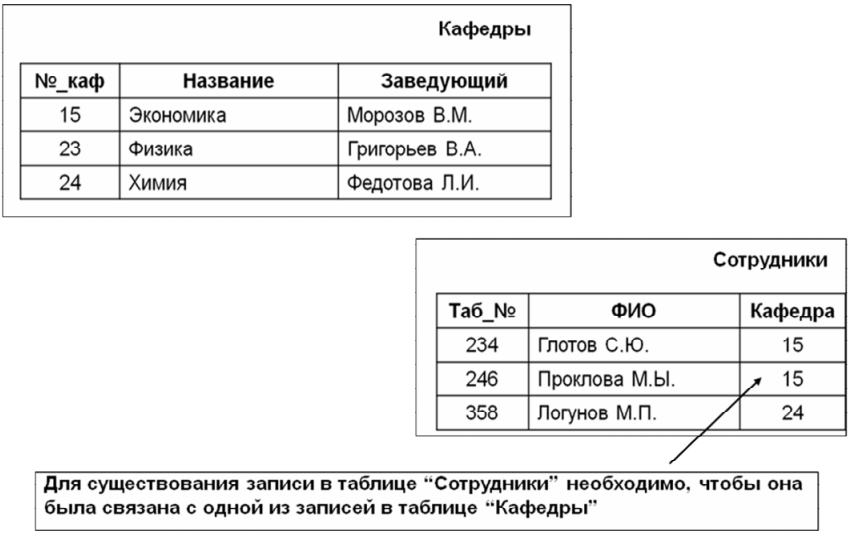


Рис. 4.3. Ограничение по существованию

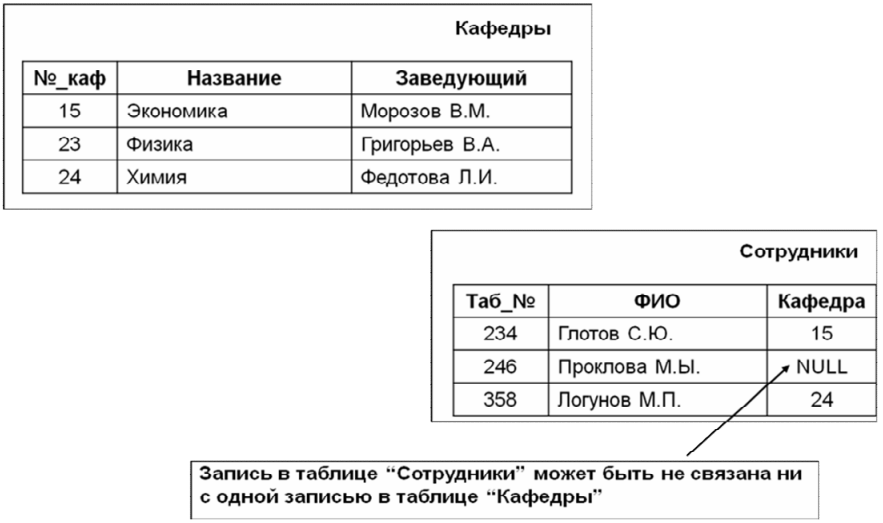


Рис. 4.4. Ограничение по связи

Установление ограничения целостности по связи и по существованию проиллюстрировано примерами, приведенными на рис. 4.3 и рис. 4.4.

Иногда в БД реализуется так называемая обратная связь по существованию, когда запись в основной таблице не может существовать без связанных с ней записей в подчиненной таблице. Такой вид ограничений целостности является более сложным с точки зрения его контроля, так как при удалении записи из подчиненной таблицы нужно проверять, есть ли в ней еще записи с таким же значением внешнего ключа.

Кроме того, в ряде случаев имеются ограничения *целостности кардинальности связи*: число элементов в связи должно находиться в некотором диапазоне – закрытом или открытом. Например, одной записи в главной таблице должно соответствовать не меньше определенного количества записей в подчиненной таблице.

Существуют также ограничения целостности, задаваемые на уровне более чем двух таблиц. Эти ограничения поддерживают логическую согласованность между данными, хранимыми во взаимосвязанных таблицах.

5. Ограничения *целостности алгоритмических зависимостей*. Если в БД имеются производные данные, которые получаются в результате каких-то операций, выполненных над другими данными (исходные данные), то изменение производных данных либо должно быть следствием изменения исходных данных, либо вообще запрещено.

6. *Запрет на обновления*. Этот запрет может относиться к любому объекту – атрибуту, строке или таблице. Так, во многих СУБД этот запрет распространяется на значения первичных ключей.

Ограничения целостности бывают одномоментными и отложенными. Отложенные ограничения в процессе выполнения каких-либо действий не соблюдаются, но должны быть соблюдены после их завершения. Примером отложенных ограничений могут являться транзакции – совокупности действий, переводящих БД из одного согласованного состояния в другое согласованное состояние. В процессе выполнения транзакции БД может какое-то время нахо-

диться в несогласованном состоянии, однако после завершения транзакции согласованное состояние должно быть восстановлено.

По режиму проверки корректности БД проверка на ограничение целостности может выполняться в момент осуществления операций над данными (оперативный режим) или независимо в заданный момент времени (аудит БД).

Ограничение целостности может распространяться не только на данные, но и на служебную информацию. В частности, в реляционных СУБД это означает поддержание соответствия между индексными файлами и соответствующими им индексируемыми файлами БД.

Существует также понятие информационной целостности банка данных, которое заключается в обеспечении согласованности совместного функционирования всех его компонентов, к которым относятся файлы БД, файлы прикладных программ, форматы ввода-вывода, отчеты и т. д.

В различных СУБД имеются разные возможности поддержки ограничений целостности. Если какое-либо ограничение целостности не поддерживается автоматически конкретной СУБД, то эту задачу должен решать администратор БД.

Для поддержки целостности может использоваться механизм триггеров – действий, активирующихся при возникновении определенного события.

Ряд ограничений целостности следует непосредственно из описания предметной области в рамках *ER*-модели.

1. Ограничение на уникальность. Ключи таблиц являются уникальными идентификаторами.

2. Между первичными ключами (уникальными идентификаторами) и другими атрибутами имеются функциональные зависимости.

3. При наличии связи между сущностями могут присутствовать ограничения по связи. Тип связи и класс ее принадлежности определяет ограничение целостности на связь между сущностями.

4. Для статических свойств сущности можно устанавливать запрет на обновление, Если свойством является некоторое условие, то значения атрибута могут быть неопределенными.

Подводя итог, необходимо подчеркнуть, что при проектировании БД нужно определить все ограничения целостности, которые следуют как из специфики предметной области, так и из особенностей используемых прикладных программ, и разработать механизмы их поддержания.

4.2. Структура SQL

Широкое развитие информационных систем и связанная с этим унифицированность информационного пространства привело к необходимости создания стандартного языка, который мог бы использоваться в большом количестве различных видов компьютерных сред. Этот язык должен позволять пользователям, владеющим навыками использования одного и того же набора команд, использовать их для создания, нахождения, изменения и передачи информации, причем независимо от того, работают ли они на персональном компьютере, сетевой рабочей станции, или на универсальной ЭВМ.

Таким стандартным языком стал язык структурированных запросов SQL (сокращенно от Structured Query Language). Язык SQL предназначен для манипулирования данными в реляционных БД, определения структуры БД и управления правами доступа к данным в многопользовательской среде. В настоящее время SQL реализован практически во всех коммерческих реляционных СУБД, все фирмы провозглашают соответствие своей реализации стандарту SQL, и на самом деле реализованные диалекты SQL очень близки (хотя и не полностью совпадают).

В стандарт SQL в качестве составных частей входят язык определения данных (Data Definition Language, DDL), язык манипулирования данными (Data Manipulation Language, DML) и язык управления данными (Data Control Language, DCL).

Язык SQL ориентирован прежде всего на групповую обработку данных. Операторы этого языка представляют пользователю информацию в табличном формате. При этом зачастую возникают конфликты между SQL и традиционными языками программирования. Это привело к разработке встроенного SQL, который позво-

ляет использовать операторы SQL в программах, написанных на традиционных языках программирования.

Существуют статический и динамический встроенный SQL.

При использовании *статического* SQL в тексте программы происходят вызовы языка SQL, которые далее включаются в выполняемый модуль после компиляции. Изменения в вызываемых функциях при этом могут происходить на уровне отдельных параметров вызовов с помощью переменных данного языка программирования.

При использовании *динамического* SQL происходит динамическое построение вызовов SQL-функций с дальнейшим обращением к данным в ходе выполнения программы. Динамический SQL применяется, как правило, когда в используемом приложении заранее неизвестен вид SQL-запроса.

Язык определения данных используется для создания и изменения структуры БД и ее составных частей – таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур. Основными его операторами являются операторы CREATE <DATABASE, TABLE, VIEW, INDEX, TRIGGER, PROCEDURE> (создать БД, таблицу БД, вид, индекс, триггер, процедуру), ALTER <> (изменить БД и т. д.), DROP <> (удалить БД и т. д.).

Оператор создания таблицы имеет следующий вид:

CREATE TABLE <имя таблицы>

(<имя столбца> <тип данных> [NOT NULL, PRIMARY KEY]

[,<имя столбца> <тип данных> [NOT NULL], [UNIQUE]]…)

Обязательными операндами в этой конструкции являются имя таблицы и имя хотя бы одного столбца с указанием типа данных для значений этого столбца. Для отдельных столбцов могут указываться дополнительные параметры, определяющие правила контроля вводимых в них значений – определение первичных ключей, уникальности значения, определенных или неопределенных значений, значений по умолчанию и т. д.

Оператор изменения структуры таблицы имеет следующий формат:

ALTER TABLE <имя таблицы>

({ADD, MODIFY, DROP} <имя столбца> [<тип данных>]
[NOT NULL]
[,{ADD, MODIFY, DROP} <имя столбца> [<тип данных>]
[NOT NULL]]…)

Операнды ADD, MODIFY, DROP используются для добавления, изменения и удаления одного или нескольких столбцов.

Оператор удаления таблицы записывается следующим образом:

DROP TABLE <имя таблицы>

Оператор создания индекса используется для создания индекса для одного или нескольких столбцов данной таблицы для целей выполнения запросов и поиска данных:

CREATE [UNIQUE] INDEX <имя индекса>
ON <имя таблицы>
(<имя столбца> [ASC/DESC]
[,<имя столбца> [ASC/DESC]…)

Операнды ASC и DESC используются для задания автоматической сортировки значений в столбцах по возрастанию или по убыванию соответственно.

Язык манипулирования данными используется, как это следует из его названия, для манипулирования данными в таблицах БД. Он состоит из 4 основных операторов: SELECT (выбрать), INSERT (вставить), UPDATE (обновить), DELETE (удалить).

Оператор выборки записей имеет вид:

SELECT [ALL/DISTINCT]
<список данных>
FROM <список таблиц>
[WHERE <условие выборки>]
[GROUP BY <имя столбца> [, <имя столбца>…]
[HAVING <условие поиска>]
[ORDER BY <спецификация> [, <спецификация>]…]

Оператор выборки записей является одним из наиболее важных операторов SQL и имеет большие функциональные возможности.

В результате выполнения операции SELECT происходит выборка данных из одной или из нескольких таблиц, которые перечисляются в списке после операнда FROM. Эти данные представ-

ляются в виде таблицы, которая может иметь (ALL) или не иметь (DISTINCT) повторяющиеся строки.

Операнд WHERE задает условия поиска, которые записываются в виде логического выражения – в него могут входить названия столбцов, арифметические операции и операции сравнения, логические условия и специальные функции.

Операнд GROUP BY используется для выделения в результирующем множестве записей группы. Под группой понимаются записи с одинаковыми значениями в тех столбцах, которые перечислены после этого операнда. Группы часто используются в логических выражениях, а также для проведения вычислений над самими группами.

Операнд HAVING используется для дополнительной селекции записей при определении списков групп. Правила его использования такие же, что и для операнда WHERE.

Для упорядочения записей применяется операнд ORDER BY.

Еще одной операцией, где используется оператор SELECT, является вложенный запрос: результаты выполнения операции SELECT используются в логическом выражении условия WHERE еще одним оператором SELECT.

Оператор изменения записей:

UPDATE <имя таблицы>
SET <имя столбца> = {<выражение>. NULL}
[, SET <имя столбца> = {<выражение>, NULL}…]
[WHERE <условие>]

Операнд SET определяет список столбцов таблицы, в которые вносятся изменения, определенные логическими или арифметическими выражениями в условии операнда WHERE. Новые значения в столбцах могут быть и пустыми (NULL).

Оператор добавления новых записей:

INSERT INTO <название таблицы>
[(<список столбцов>)]
VALUES (<список значений>)

или

INSERT INTO <название таблицы>
[(<список столбцов>)]
<предложение SELECT>

В первом случае добавляются новые записи с определенными значениями в столбцах. Во втором случае в таблицу вводятся новые записи, выбранные из другой таблицы оператором SELECT.

Оператор удаления записей:

DELETE FROM <название таблицы>
[WHERE <условие>]

Из таблицы удаляются записи, удовлетворяющие условию, определенному операндом WHERE. В случае отсутствие операнда WHERE из таблицы удаляются все записи.

Язык управления данными используется для управления правами доступа к данным и выполнения процедур в многопользовательской среде. Более точно его можно назвать “язык управления доступом”. Он состоит из двух основных операторов GRANT (дать права) и REVOKE (забрать права).

В любом случае SQL работает с данными, имеющими вид таблиц. Имеется два вида таблиц: базовые таблицы, т. е. таблицы, определенные и описанные на языке описания данных, и производные таблицы, получаемые из нескольких таблиц путем выполнения запроса и определенные на языке манипулирования данными.

В SQL предусмотрен механизм декларативного задания ограничения целостности, которое задается преимущественно на уровне описания таблицы данных. При этом могут быть реализованы ограничения целостности различных уровней, которые задаются посредством определенных языковых конструкций при описании таблицы.

При ограничении уникальности устанавливается требование, согласно которому две строки в таблице не могут быть иметь одинаковых значений в данном столбце или совокупности столбцов. Данный тип ограничения может быть использован для определения возможных ключей таблицы, один из которых выбирается в качестве первичного (PRIMARY KEY), а остальные определяются с помощью использования оператора UNIQUE. Если первичный ключ является простым, т. е. состоящим из одного атрибута, то ограничение PRIMARY KEY может быть задано или непосредственно при описании этого атрибута или в самом конце описания всей таблицы. Если же первичный ключ является составным, то возможен только второй способ. Ограничение UNIQUE устанавливается

почти так же как и ограничение PRIMARY KEY с тем отличием, что оно допускает возможность неопределенных NULL значений.

Для определения ограничения ссылок используются ключевые слова FOREIGN KEY и REFERENCES. Первое из них используется для определения внешних ключей данной (подчиненной, дочерней) таблицы, а второе – для определения внешней (главной, родительской) таблицы и ее полей, составляющих первичный ключ. В родительской таблице для атрибутов, на которые происходит ссылка, всегда должно быть установлено ограничение на их уникальность.

При задании проверочного ограничения выполняется проверка (CHECK) некоторого условия для значения данных в каждой строке таблицы – указывается предикат, который использует значения атрибутов для вычисления некоторого значения. Предикат может принимать значения TRUE, FALSE и UNKNOWN. Ограничение CHECK будет нарушено, если предикат принимает значение FALSE.

Кроме того, существует ограничение на определенность значения NOT NULL. Если это ограничение задано для некоторого столбца таблицы, то это означает, что для данных, представленных в этом столбце, либо должны быть заданы значения по умолчанию, либо при любой модификации этих данных значение не должно стать неопределенным.

Для задания ограничений используется оператор CONSTRAINT. Конструкция этого оператора имеет следующий вид:

CONSTRAINT [идентификатор ограничения] <имя ограничения> <выражение>

Существуют следующие идентификаторы ограничений:

PK – ограничение на первичный ключ;

FK – ограничение на внешний ключ;

U – ограничение на уникальность значения;

DF – ограничение на значение по умолчанию;

CK – ограничение на проверочное условие CHECK.

Вот как, например, может выглядеть код создания двух связанных таблиц.

```
CREATE TABLE Кафедра
(
№_каф smallint UNIQUE, NOT NULL,
Название char(30), UNIQUE, NOT NULL,
Заведующий char(60), NOT NULL,
CONSTRAINT PK_Кафедра PRIMARY KEY (№_каф),
                                    )
CREATE TABLE Сотрудники
(
(Таб_№ smallint PRIMARY KEY, UNIQUE, NOT NULL,
ФИО char(60), UNIQUE, NOT NULL,
№_каф smallint, NULL,
Оклад Integer, NOT NULL,
Раб_телефон char(17), NULL,
Дом_телефон char(17), NULL,
CONSTRAINT FK_Кафедра FOREIGN KEY (№_каф) REFERENCES Кафедра (№_каф),
CONSTRAINT DF_Оклад DEFAULT(2000)
CONSTRAINT CK_Телефон CHECK(Раб_телефон IS NOT NULL OR Дом_телефон IS NOT NULL),
                                                    )
```

В SQL существуют понятия неотложенного и отложенного ограничения целостности. Как правило, по умолчанию ограничения целостности определяются как неотложенные, что означает, что проверка на целостность осуществляется при выполнении каждого оператора SQL.

В SQL поддерживается большое количество типов данных (табл. 4.1), среди которых можно выделить следующие.

1. *Числовые типы* данных. Типы данных INT и SMALLINT задают целые числа. Эти типы удобно задавать, например, для идентификаторов, количества, возраста и т. д. Типы данных NUMERIC и DECIMAL определяют десятичные числа с фиксированным количеством знаков после запятой. Их можно использовать для хранения результатов арифметических операций. Для определения данных с переменным количеством знаков после запятой используются типы REAL и FLOAT. Диапазон допустимых значений дан-

ных, определяемых этими типами, существенно больше по сравнению с десятичными числами.

2. *Символьные типы* данных. Типы CHAR и VARCHAR определяют строки постоянной и переменной длины соответственно. Тип данных TEXT (длинный текст) используется для хранения больших документов.

Таблица 4.1. Типы данных SQL

Тип данных	**Описание**
INT, SMALLINT	Целые числа. Удобно задавать для идентификаторов, количества, возраста
NUMERIC, DECIMAL	Десятичные числа с фиксированным количеством знаков после запятой. Можно использовать для хранения результатов арифметических операций
REAL, FLOAT	Числа с плавающей запятой. Диапазон допустимых значений больше по сравнению с десятичными числами
CHAR	Символьные строки постоянной длины. Используются для хранений фамилий, названий, адресов
VARCHAR	Символьные строки переменной длины.
MONEY, SMALLMONEY	Хранение денежных величин. Можно снабжать указателем на вид валюты
DATETIME, SMALLDATETIME	Хранение данных в формате дата/время.
BIT	Логический (булев) тип данных. Значениями такого типа могут быть Истина (1) и Ложь (0)
TEXT	Так называемый длинный текст. Используется для хранения документов
BINARY, VARBINARY, IMAGE	Неструктурированные типы данных. Используются, в частности, для хранения видеоизображений

3. *Денежные типы* данных MONEY и SMALLMONEY позволяют хранить денежные величины с заданием при необходимости признака валюты.

4. *Данные типа время/дата* DATETIME и SMALLDATETIME. Такой тип данных удобно задавать при необходимости обработки временных интервалов, например, для сравнения текущей даты с какой-либо фиксированной датой.

5. *Логические типы* данных. Тип данных BIT определяет так называемые булевы величины, которые могут принимать два значения – Истина и Ложь.

6. *Неструктурированные типы* данных BINARY, VARBINARY, IMAGE применяются для определения таких данных, как графические объекты, видеоизображения и других неструктурированных байтовых потоков.

Таблица 4.2. Встроенные функции SQL

Функция	**Назначение**
ABS(N)	Вычисление модуля числа N
SIGN(N)	Определение знака числа N
RAND	Генератор случайного числа в диапазоне от 0 до 1
ROUND(N,δ)	Округление числа N с точностью δ
POWER(N,M)	Возведение числа N в степень M
SIN(N), COS(N)	Тригонометрические функции
EXP(N)	Экспонента числа N
LOG(N)	Натуральный логарифм числа N
LEN(S)	Вычисление длины строки символов S
LTRIM(S), RTRIM(S)	Удаление пробелов в начале и в конце строки S
LEFT(S,i)	Копирование i символов строки S начиная с левого символа
RIGHT(S,i)	Копирование i символов строки S начиная с правого символа
SUBSTRING(S,i,j)	Копирование подстроки длины j строки S начиная с i-го символа
STR(N)	Преобразование числа N в символьный тип
LOWER(S)	Перевод строки S в нижний регистр
UPPER(S)	Перевод строки S в верхний регистр

SQL содержит большое количество встроенных функций (табл. 4.2), позволяющих обрабатывать данные. Среди этих функций наиболее широко используемыми являются математические, статистические функции, функции для работы с данными время/дата, символьными данными, системные функции, функции конфигурирования и функции, обеспечивающие безопасность системы.

4.3. Запросы на выборку

Запросы используются для получения пользователем информации, содержащейся в БД, в удобном для него виде. Результат запроса отображается для пользователя в виде таблицы, содержащей требуемую этим пользователем информацию. Запросы делятся на поисковые и корректирующие. Первый тип запросов используется только лишь для отображения данных, второй применяется для возможности внесения изменений в данные – их модификацию.

Основным оператором для отбора информации из БД является оператор SELECT.

Формат этого оператора имеет следующий вид:

SELECT [DISTINCT]

{<функция агрегирования>/<выражение для вычисления значения>

[AS <имя столбца>]}

FROM {{<имя таблицы> [AS] [<имя корреляции>].[<имя столбца>.,..]}

{подзапрос [AS] [<имя корреляции>].[<имя столбца>.,..]}

<соединенная таблица>}.,..

[WHERE <условие>]

[GROUP BY {{[<имя таблицы>/<имя корреляции>]}].[<имя столбца>}.,..}]

[HAVING <условие>]

[UNION/INTERSECT/EXCEPT][ALL]

[CORRESPONDING [BY (<имя столбца>.,..)]]

<оператор SELECT>/ TABLE <имя таблицы>/<конструктор значений таблицы>

[ORDER BY {{<столбец-результат> [ASC/DESC]}.,..}

Операнды SELECT, FROM, WHERE, GROUP BY, HAVING и ORDER BY должны записываться в той последовательности, в которой они перечислены.

SELECT является ключевым словом, которое для СУБД означает, что последующая команда является запросом. Операнд FROM должен обязательно присутствовать в каждом запросе. Все остальные операнды являются необязательными.

Самый простой вариант запроса соответствует случаю декартова произведения таблиц T1 и T2:

```
SELECT *
FROM T1, T2
```

Реализация запроса в виде:

```
SELECT T1.A, T2.B
```

соответствует проекции декартова произведения двух таблиц на столбцы A и B исходных таблиц, однако в отличие от соответствующей операции реляционной алгебры дубликаты всех строк при этом сохраняются.

Операнд SELECT служит для определения столбцов таблицы, которая будет получена при выполнении запроса. Столбец этой таблицы может иметь название, как совпадающие с названием столбца базовой таблицы, так и собственное определяемое пользователем. Если в результате запроса данные отбираются из нескольких таблиц, в которых имеются столбцы с одинаковым названием, то сначала указывается имя таблицы, содержащей данный столбец, и далее через точку имя столбца.

Операнд AS используется, во-первых, для определения вычисляемых столбцов, значения которых задаются в результате вычисления выражений, во-вторых, в тех случаях, когда название столбца, являющегося результатом запроса, отличается от названия столбца таблицы БД.

Если в результате выполнения запроса результирующая таблица имеет повторяющиеся строки, то можно использовать операнд DISTINCT для избежания дублирования таких строк.

Запросы могут содержать следующие функции агрегирования.

1. COUNT – функция подсчета. Имеется два варианта использования этой функции. В первом случае функция подсчитывает количество строк в таблице или в группе, если используется в операнде

GROUP BY. Смысл такого использования заключается в том, что результат подсчета не зависит от того, имеются ли в столбцах значения NULL и указан ли параметр DISTINCT. Во втором случае результат зависит от значений этих параметров.

2. SUM – суммирование значений в столбце.

3. MAX, MIN, AVERAGE – вычисление максимального, минимального и среднего значений.

Операнд FROM используется для указания списка таблиц, из которых будут выбираться данные при выполнении запроса. Если в списке операнда FROM перечисляется несколько таблиц, то эти таблицы считаются соединяемыми. Причем если не указывать в явном виде типа соединения, то считается по умолчанию, что каждая строка первой таблицы соединяется с каждой строкой второй таблицы. Такое соединение называется *перекрестным* и оно соответствует операции прямого декартова произведения. Пример перекрестного соединения приведен на рис. 4.5.

Таблица 1

Таб. №	ФИО
18/01	Иванов Д.Б.
14/02	Козлов А.Ю.
22/03	Петренко С.М.

Таблица 2

ФИО	Предмет
Иванов Д.Б.	Базы данных
Иванов Д.Б.	Паскаль
Козлов А.Ю.	Пение
Глотов В.В.	Физика

Результат – прямое декартово произведение

Таб. №	Таблица 1	Таблица 2	Предмет
18/01	Иванов Д.Б.	Иванов Д.Б.	Базы данных
18/01	Иванов Д.Б.	Иванов Д.Б.	Паскаль
18/01	Иванов Д.Б.	Козлов А.Ю.	Пение
18/01	Иванов Д.Б.	Глотов В.В.	Физика
14/02	Козлов А.Ю.	Иванов Д.Б.	Базы данных
14/02	Козлов А.Ю.	Иванов Д.Б.	Паскаль
14/02	Козлов А.Ю.	Козлов А.Ю.	Пение
14/02	Козлов А.Ю.	Глотов В.В.	Физика
22/03	Петренко С.М.	Иванов Д.Б.	Базы данных
22/03	Петренко С.М.	Иванов Д.Б.	Паскаль
22/03	Петренко С.М.	Козлов А.Ю.	Пение
22/03	Петренко С.М.	Глотов В.В.	Физика

Рис. 4.5. Перекрестное соединение

Для указания в явном виде типа соединения таблиц используется встроенный операнд JOIN.

Наиболее часто используется тип соединения INNER JOIN – внутреннее соединение. При таком типе соединения таблица будет включать только те строки, для которых имеются соответствующие друг другу значения связанных полей в двух таблицах.

Пусть, например, таблица 1 на рис. 4.5 является списком только штатных преподавателей вуза, а таблица 2 содержит информацию об учебной нагрузке в текущем семестре, причем в этой таблице представлены данные обо всех преподавателях, включая совместителей.

Таб. №	ФИО	Предмет
18/01	Иванов Д.Б.	Базы данных
18/01	Иванов Д.Б.	Паскаль
14/02	Козлов А.Ю.	Пение

Рис. 4.6. Результат внутреннего соединения таблиц

В случае внутреннего соединения получится результат, показанный на рис. 4.6.

Как видно, в итоге получена информация о том, какие предметы ведут в текущем семестре только штатные преподаватели.

Если указан тип соединения LEFT JOIN (левое внешнее соединение), то в соединенную таблицу попадают все строки из первой таблицы и только те строки из второй таблицы, для которых в первой таблице имеются поля связи.

Таб. №	ФИО	Предмет
18/01	Иванов Д.Б.	Базы данных
18/01	Иванов Д.Б.	Паскаль
14/02	Козлов А.Ю.	Пение
22/03	Петренко С.М.	Null

Рис. 4.7. Результат левого внешнего соединения таблиц

В рассмотренном выше примере использование левого внешнего соединения позволило получить информацию об учебной нагрузке штатных преподавателей (рис. 4.7).

RIGHT JOIN (правое внешнее соединение) является обратным по сравнению с левым внешним соединением – в соединенной таблице имеются все строки из второй таблицы и только те строки из первой таблицы, для которых во второй таблице имеются поля связи.

Таб. №	ФИО	Предмет
18/01	Иванов Д.Б.	Базы данных
18/01	Иванов Д.Б.	Паскаль
14/02	Козлов А.Ю.	Пение
Null	Глотов В.В.	Физика

Рис. 4.8. Результат правого внешнего соединения таблиц

Для нашего примера правое внешнее соединение позволило получить список всех предметов, изучаемых в текущем семестре, с указанием преподавателей по каждому предмету с их штатной принадлежностью (рис. 4.8).

Кроме перечисленных типов соединения, которые поддерживаются большинством СУБД, в ряде систем присутствуют типы соединений FULL JOIN и UNION JOIN, которые фактически являются объединением и пересечением типов LEFT JOIN и RIGHT JOIN соответственно.

Операнд WHERE используется для отбора записей в таблицах БД в соответствии с указанными условиями поиска. Этих условий может быть несколько и в них могут входить операторы сравнения и логические функции. Допустимо также использование специальных предикатов.

1. Интервальный предикат используется для задания диапазона значений:

WHERE [NOT] <выражение> BERWEEN <нижнее значение выражения> AND <верхнее значение выражения>.

2. Предикат IN:

WHERE [NOT] <выражение> [NOT] IN (<список значений>/<подзапрос>)

3. Предикат проверки на неопределенное значение:

WHERE <значение> IS [NOT] NULL

4. Предикат подобия:

WHERE <выражение 1> [NOT] LIKE <выражение 2>

Кроме того, при выполнении подзапросов в условии WHERE может быть использован операнд EXIST, который проверяет возврат подзапросом каких-либо данных:

WHERE [NOT] EXIST <подзапрос>.

Операнд GROUP BY всегда используется со встроенными агрегированными функциями и применяется для определения выходных данных, определенным образом сгруппированных. Операнд GROUP BY работает всегда на одном уровне в том смысле, что сгруппированные им данные (группу) нельзя разбивать в свою очередь на группы более низких уровней.

Кроме того, если применить оператор SELECT к сгруппированным данным, то действие этого оператора применяется к каждой группе, а не к каждой строке, как это происходит в обычной ситуации. Это означает, что каждое выражение оператора SELECT становится единственным для всей группы.

Операнд HAVING часто используется вместе с операндом GROUP BY, причем условие, записанное в этом операнде, имеет то же самое значение для групп, что и условие, записанное в операнде WHERE для строк. Выражение оператора HAVING должно принимать единственное значение для всей группы.

Для упорядочения данных, получаемых в результате запроса, применяется операнд ORDER BY. В списке этого операнда указываются столбцы, которые сортируются в соответствии со значениями в этих столбцах. В этом списке могут быть как названия столбцов, так и номера столбцов (целые числа). При этом столбцы всегда нумеруются целым числом в случае, если они являются вычисляемыми, либо получаются в результате использования операнда UNION. Если в списке операнда UNION указывается список столбцов, то в этом случае происходит упорядочение по составному ключу.

Отдельного рассмотрения требуют запросы, обращающиеся к данным, содержащимся в нескольких таблицах. Эти запросы могут быть реализованы несколькими различными способами.

1. В условии WHERE могут быть заданы в явном виде правила соединения таблиц.

2. Могут быть использованы вложенные запросы.

3. Можно использовать различные конструкции с использованием операнда JOIN.

Рассмотренные выше конструкции относились к поисковым запросам. В корректирующих запросах основными операторами являются операторы INSERT, UPDATE и DELETE.

Оператор INSERT предназначен для включения в таблицу новых строк и имеет следующий формат:

INSERT INTO <название таблицы>
[<название столбца>.,..] <условие запроса>/<конструктор значений>
{DEFAULT VALUES}

Значения должны вводиться в столбцы, перечисленные в списке, в том порядке, в котором они указаны, в противном случае значения должны вводиться во все столбцы новых строк.

Оператор UPDATE используется для корректировки содержания таблицы:

UPDATE <название таблицы> SET <название столбца> = <новое значение> [,<название столбца > = < новое значение >…]
[WHERE <условие>]

При использовании этого оператора можно изменять значение в столбце для всех записей таблицы (при отсутствии условия WHERE) или только для тех записей, которые удовлетворяют условию WHERE.

Оператор DELETE используется для удаления строк из таблицы:

DELETE
FROM <название таблицы>
[WHERE <условие>]

Если условие WHERE отсутствует, то удаляются все строки таблицы.

Приведем ряд примеров использования возможностей SQL для формирования запросов.

Преподаватель

Таб_№	ФИО	Группа	Предмет
118	Зуйков В.М.	12-01	Паскаль
119	Королев С.Ш.	14-03	Базы данных
125	Мамедов В.Ч.	05-02	Физика
153	Ковшов Г.Д.	15-04	Математика
579	Прошина В.А.	NULL	NULL

Результат_сессии

№_зач	Идентификатор	ФИО	Группа	Предмет	Дата	Оценка
12/1	1324	Гуров П.В.	14-03	Паскаль	12.01.08	2
13/1	0653	Гуров П.В.	14-03	Математика	15.01.08	5
27/2	1283	Миронов Р.Е.	12-01	Базы данных	05.01.08	3
43/2	1004	Соболева Я.Д.	15-04	Физика	11.01.08	NULL
56/1	1782	Гуров П.В.	14-03	Физика	14.01.08	2
66/3	0073	Кулаков Ф.Г.	14-03	Паскаль	14.01.08	2

Рис. 4.9. Таблицы базы данных

Рассмотрим в качестве примера две связанные таблицы, изображенные на рис. 4.9. В таблице “Преподаватель” содержатся сведения о предметах, занятия по которым ведут перечисленные в ней преподаватели, с указанием групп, обучающихся по данным предметам. В таблице “Результат_сессии” заносятся данные как об удачных, так и неудачных попытках студентов сдать экзамены. Значение “NULL” в последнем столбце этой таблицы означает, что студент не явился на сдачу экзамена. Таблицы связаны между собой по общему атрибуту “Предмет”.

Начнем с простых запросов.

Получим сначала список групп, в которых должны быть проведены экзамены:

```
SELECT DISTINCT Группы
FROM Результат_сессии
```

Получим:

Группа
14-03
12-01
15-04

Теперь получим список студентов, сдавших математику на оценку “отлично”:

SELECT ФИО, Группа
FROM Результат_сессии
WHERE Предмет = ‘Математика’ AND Оценка = ‘5’
Результат таков:

ФИО	**Группа**
Гуров П.В.	14-03

Для составления расписания экзаменов преподавателей составляем следующий запрос:

SELECT Преподаватель.ФИО, Результат_сессии.Предмет, Дата
FROM Преподаватель, Результат_сессии
WHERE Преподаватель.Предмет = Результат_сессии. Предмет
В итоге:

Фамилия	**Предмет**	**Дата**
Зуйков В.М	Паскаль	12.01.08
Королев С.Ш.	Базы данных	05.01.08
Мамедов В.Ч.	Физика	11.01.08
Мамедов В.Ч.	Физика	14.01.08
Ковшов Г.Д.	Математика	15.01.08

Выведем список студентов, имеющих несколько “двоек”:

SELECT ФИО
FROM Результат_сессии a, Результат_сессии b
WHERE a.Предмет <> b.Предмет AND a.Оценка = ‘2’
AND b.Оценка = ‘2’
Результат:

ФИО
Гуров П.В.

Обратите внимание, что в тексте запроса были использованы псевдонимы a и b, что связано с необходимостью проводить вычисления с несколькими экземплярами одной таблицы.

Выведем список студентов, не сдававших в срок экзамены по каким-либо предметам, с указанием этих предметов:

SELECT ФИО, Предмет
FROM Результат_сессии
WHERE Оценка IS NULL
Результат:

ФИО	**Группа**
Соболева Я.Д.	Физика

Теперь составим несколько запросов с использованием агрегированных функций.

Получим информацию о том, сколько студентов должно сдавать экзамен по каждому предмету:

SELECT Предмет, COUNT(*)
FROM Результат_сессии
GROUP BY Предмет
Результат:

Предмет	**COUNT(*)**
Паскаль	2

Базы данных	1
Физика	2
Математика	1

Далее посмотрим, сколько студентов уже сдали экзамены по каждому из предметов:

```
SELECT Предмет, COUNT(*)
FROM Результат_сессии
WHERE Оценка IS NOT NULL
GROUP BY Предмет
```

Результат:

Предмет	**COUNT(*)**
Паскаль	2
Базы данных	1
Физика	1
Математика	1

Вычислим средний балл для каждой группы по каждому предмету:

```
SELECT Группа, Предмет, AVG(Предмет)
FROM Результат_сессии
WHERE Оценка IS NOT NULL
GROUP BY Группа, Предмет
```

Результат:

Группа	**Предмет**	**AVG (Предмет)**
14-03	Паскаль	2
14-03	Математика	5

12-01	Базы данных	3
15-04	Физика	2

Получим список групп, в которых по какому-либо предмету имеется несколько двоек:

```
SELECT Группа, Предмет FROM Результат_сессии
WHERE Оценка = ‘2’
GROUP BY Группа
HAVING COUNT(Группа) > 1
```

Результат:

Группа	**Предмет**
14-03	Паскаль

Получим информацию о том, по каким предметам экзамены должны быть проведены в определенный срок:

```
SELECT Предмет, Дата
FROM Результат_сессии
WHERE Дата BETWEEN 10.01.08 AND 14.01.08
```

Результат:

Предмет	**Дата**
Паскаль	12.01.08
Физика	11.01.08
Физика	14.01.08
Паскаль	14.01.08

В этом запросе была использована конструкция BETWEEN … AND, которая позволила выбирать данные из указанного диапазона значений.

Операнд IN используется для указания атрибута, в множестве значений которого необходимо осуществлять поиск. С помощью

этого операнда получим сведения о том, как данная группа сдала экзамены по определенному предмету:

```
SELECT ФИО, ОЦЕНКА
FROM Результат_сессии
WHERE Группа IN (‘14-03’) AND Предмет IN (‘Паскаль’)
```

Результат:

ФИО	Оценка
Гуров П.В.	2
Кулаков Ф.Г.	2

Наряду с операндом IN может быть использован оператор LIKE. Получим данные о результатах экзамена по физике:

```
SELECT Дата, Оценка
FROM Результат_сессии
WHERE Предмет LIKE ‘Физика’
```

Результат:

Дата	Оценка
11.01.08	NULL
14.01.08	2

Рассмотрим другой пример (рис. 4.10): база данных содержит сведения о штатных сотрудниках кафедр вуза. Определим кафедры, на которых имеется не менее трех сотрудников:

```
SELECT Каф_№, COUNT(ФИО)
AS Число сотрудников
FROM Сотрудник
GROUP BY Каф_№
HAVING COUNT(ФИО) > 2
```

Результат:

Каф_№	Число сотрудников
1	3
4	3

Кафедра

Каф_№	Заведующий	Телефон
1	Сайков А.	231-72-14
3	Чумаков В.	231-34-17
4	Громов О.	231-45-54
9	Борисов М.	231-45-87
12	Приходько Й.	231-22-65

Сотрудник

Таб_№	ФИО	Кафедра	Должность	Оклад	Премия
241	Абрамов С.	9	Доцент	3000	7500
256	Павлов Р.	4	Профессор	5000	7500
257	Прудников А.	4	Доцент	3000	3000
258	Козлович Ж.	1	Доцент	3000	NULL
287	Бажков Н.	3	Профессор	5000	2000
289	Дорохов А.	4	Ст. преп.	2000	3200
301	Быков В.	1	Доцент	3000	4500
302	Окороков К.	12	Профессор	5000	NULL
310	Шкловский Я.	1	Доцент	3000	4200

Рис. 4.10. Таблицы базы данных

Использование в этом запросе операнда AS позволило переименовать название столбца в виртуальной таблице результата запроса.

Получим данные о том, кто из сотрудников кафедр получил премии:

```
SELECT ФИО, Премия
FROM Сотрудник
WHERE Премия IS NOT NULL
ORDER BY Премия DESC
```

Результат:

ФИО	Премия
Абрамов С.	7500
Павлов Р.	7500
Быков В.	4500
Шкловский Я.	4200

Дорохов А.	3200
Прудников А.	3000
Бажков Н.	2000

В этом запросе для сортировки сумм премий по убыванию был использован операнд DESC.

Последние два примера иллюстрируют вложенные запросы.

Сначала выведем список сотрудников, получивших премии:

```
SELECT ФИО
FROM Сотрудник
WHERE Таб_№ = ANY
(SELECT Таб_№ FROM Сотрудник WHERE Премия IS NOT NULL)
```

Результат:

ФИО
Абрамов С.
Павлов Р.
Прудников А.
Бажков Н.
Дорохов А.
Быков В.
Шкловский Я.

Операнд ANY в этом запросе позволяет выбрать записи, удовлетворяющие сравнению с записью вложенного запроса.

Теперь выведем список сотрудников, не получивших премии:

```
SELECT ФИО
FROM Сотрудник
WHERE Таб_№ = NOT IN
(SELECT Таб_№ FROM Сотрудник WHERE Премия IS NULL)
```

Результат:

ФИО
Козлович Ж.
Окороков К.

В заключении раздела приведем несколько примеров изменения данных – добавления, изменения и удаления записей.

Добавим нового сотрудника в соответствующую таблицу, внеся значения во все поля:

```
INSERT INTO Сотрудник
VALUES (503, ‘Семенов А.’, 9, ‘Доцент’, 3000, NULL)
```

Если все поля заполнить сразу невозможно, то может быть использован, например, такой вариант:

```
INSERT INTO Сотрудник (ФИО)
VALUES (‘Григорьев Б.’)
```

Можно перевести сотрудника в новую должность с изменением его оклада:

```
UPDATE Сотрудник
SET Должность = ‘Доцент’, Оклад = 3000
WHERE Таб_№ = 289
```

В случае увольнения сотрудника удалить его из базы данных можно следующими способами:

```
DELETE Сотрудник.ФИО
FROM Сотрудник
WHERE (Сотрудник.ФИО) = ‘Дорохов А.’
```

или:

```
DELETE
FROM Сотрудник
WHERE ФИО = ‘Дорохов А.’
```

Если же, например, одна из кафедр была ликвидирована, и информацию о ней нужно удалить из обеих таблиц, то это можно сделать следующим образом:

```
DELETE
FROM Кафедра
WHERE Каф_№ = 4
```

Однако следует заметить, что удаление записей из таблицы “Сотрудник”, связанных с 4-й кафедрой, будет возможно, если между таблицами “Кафедра” и “Сотрудник” установлено каскадное удаление записей. В противном случае целостность данных нарушится и операция удаления будет отклонена.

4.4. Создание представлений и курсоров

Представлением (View) называется виртуальная таблица, которая отображает данные, получаемые из реальных таблиц БД, а также из других представлений. Представление может быть получено как из одной таблицы БД, так и из нескольких. Кроме того, в представлении допускается включение вычисляемых полей. В общем случае представление можно рассматривается как хранимый запрос на выборку.

В отличие от реальной таблицы представление действительно не существует в БД, хотя все зафиксированные в нем данные реально существуют в БД в разных ее таблицах. Эти данные компонуются в удобном для пользователя виде с помощью запроса. При создании представление обязательно получает уникальное имя и его описание хранится в описании схемы БД. СУБД при обращении к представлению выполняет запрос, соответствующий описанию этого представления.

Оператор определения представления имеет следующий формат:

```
CREATE VIEW <имя представления>
[(<имя столбца> [, <имя столбца>]…)]
AS <оператор SELECT>
[WITH[CASCADED/LOCAL]CHECK OPTIONS]
```

Выполнение этого оператора приводит к созданию виртуальной таблицы, состав которой определяется оператором SELECT. Пользователь может работать с представлениями как с реальной таблицей, не зная даже при этом, что на самом деле он работает с виртуальной таблицей. Понятие виртуальной таблицы означает, что представление не хранится в памяти компьютера в виде физической таблицы данных – сохраняется лишь описание представления, следующее за оператором AS. Сама же виртуальная таблица формируется лишь в момент обращения к представлению.

При необходимости можно задавать любые имена для столбцов виртуальной таблицы, создаваемой представлением. В случае, если список имен столбцов не указывается, то каждый столбец получает имя соответствующего столбца реальной таблицы БД. В явном виде имена столбцов должны задаваться в случаях, когда какой-то столбец представления получается в результате вычисления арифметического выражения, является результатом функции, константой, т. е., другими словами, тогда, когда в реальной таблице ему не соответствует ни один столбец, чье имя он может наследовать. Имя столбца должно также задаваться в случае, когда без этого два или более столбца имели бы совпадающие имена.

Для уничтожения представления используется стандартный оператор DROP:

DROP VIEW <имя представления>

Представления могут быть использованы для достижения следующих целей.

1. Скрытие от пользователя не предназначающейся для него информации.

2. Обеспечение улучшенной степени защиты данных: часть данных скрывается от пользователей в соответствии с правами доступа различных категорий пользователей.

3. Упрощение сложных запросов.

4. Предоставление пользователю дополнительной информации, не содержащейся в реальных таблицах: в этом случае значения столбцов в представлении являются вычисляемыми.

Последняя строка в операторе создания представления (WITH CHECK OPTION) означает, что при создании представления должно проверяться некоторое заданное условие.

Основными базовыми видами представлений, на основе которых строятся более сложные представления, являются горизонтальные, вертикальные, сгруппированные и объединенные представления.

Горизонтальное представление используется, главным образом, для уменьшения объема реальных таблиц и имеет вид:

```
CREATE VIEW <имя представления>
AS
SELECT *
FROM <имя таблицы>
WHERE <имя столбца> = <значение>
```

Это представление является фактически аналогом операции выборки реляционной алгебры. В результате его реализации виртуальная таблица будет содержать весь набор столбцов реальной таблицы и столько строк, сколько раз в заданном условием WHERE столбце встречается данное значение.

Вот два примера горизонтальных представлений, выполненных для БД, приведенной на рис. 4.9.

```
CREATE VIEW Кафедра_4
AS
SELECT *
FROM Сотрудник
WHERE Кафедра = 4
```

Результат:

Таб_№	ФИО	Каф.	Должность	Оклад	Премия
256	Павлов Р.	4	Профессор	5000	7500
257	Прудников А.	4	Доцент	3000	3000
289	Дорохов А.	4	Ст. преп.	2000	3200

```
CREATE VIEW Кафедра_4
AS
SELECT *
FROM Кафедра
WHERE Кафедра = 4
```

Результат:

Каф_№	Заведующий	Телефон
4	Громов О.	231-45-54

Аналогом операции проектирования является *вертикальное представление*, отображающее все строки реальной таблицы и только те столбцы, которые задаются оператором SELECT:

```
CREATE VIEW <имя представления>
[(<имя столбца> [, <имя столбца>]…)]
AS
SELECT [(<имя столбца> [, <имя столбца>]…)]
FROM <имя таблицы>
```

Ниже приведены два примера вертикальных представлений.

```
CREATE VIEW Должности
AS
SELECT ФИО, Должность
FROM Сотрудник
```

Результат:

ФИО	Должность
Абрамов С.	Доцент
Павлов Р.	Профессор
Прудников А.	Доцент
Козлович Ж.	Доцент
Бажков Н.	Профессор
Дорохов А.	Ст. преп.
Быков В.	Доцент
Окороков К.	Профессор
Шкловский Я.	Доцент

CREATE VIEW Телефоны_кафедр
AS
SELECT Каф_№, Телефон
FROM Кафедра
Результат:

Каф_№	Телефон
1	231-72-14
3	231-34-17
4	231-45-54
9	231-45-87
12	231-22-65

Сгруппированные представления содержат в себе запросы, имеющие группировку.

В них возможно использование агрегированных функций, и в дальнейшем данные этого типа представления можно использовать в других запросах. Формат такого представления имеет вид:

CREATE VIEW <имя представления>
[(<имя столбца> [, <имя столбца>]…)]
AS
SELECT [([<функция>]<имя столбца> [, <имя столбца>]…)]
FROM <имя таблицы>
GROUP BY [(<имя столбца> [, <имя столбца>]…)]

Приводим пример сгруппированного представления.

CREATE VIEW Сводные_данные
Кафедра, COUNT(*), SUM(Оклад), SUM(Премия), AVG(Оклад), AVG(Премия)
AS
SELECT Кафедра, COUNT(*), SUM(Оклад), SUM(Премия), AVG(Оклад), AVG(Премия)
FROM Сотрудник

GROUP BY Кафедра

Результат:

Кафедра	COUNT(*)	SUM (Оклад)	SUM (Премия)	AVG (Оклад)	AVG (Премия)
1	3	9000	8700	3000	2900
3	1	5000	2000	5000	2000
4	3	10000	13700	3333.33	4566.67
9	1	3000	7500	3000	7500
12	1	5000	NULL	5000	NULL

Объединенные представления используются для представления в одной виртуальной таблице данных из нескольких таблиц БД:

CREATE VIEW <имя представления>
[(<имя столбца> [, <имя столбца>]…)]
AS
SELECT [(<имя столбца> [, <имя столбца>]…)]
FROM [(<имя таблицы> [, <имя таблицы>]…)]
[WHERE <условие>]

Следующий пример приведен для базы данных, таблицы которой приведены на рис. 4.9.

CREATE VIEW Плохие_Оценки
Преподаватель, Предмет, Студент, Оценка
AS
SELECT Преподаватель.ФИО, Преподаватель.Предмет, Результат_сессии.ФИО,
Оценка
FROM Преподаватель, Результат_сессии
WHERE Преподаватель.Предмет = Результат_сессии.Предмет AND Оценка = 2

Результат:

Преподаватель	Предмет	Студент	Оценка
Зуйков В.М	Паскаль	Гуров П.В.	2
Мамедов В.Ч.	Физика	Гуров П.В.	2

Зуйков В.М	Паскаль	Кулаков Ф.Г.	2

Несмотря на то, что зачастую представления можно использовать в операциях реляционного исчисления как реальные отношения, имеются ограничения на операции модификации данных, содержащихся в представлениях. Обновление данных через представления (корректирующий запрос) возможно лишь в тех случаях, когда каждая строка (столбец) представления может быть однозначно сопоставлена строке (столбцу) реальной таблицы данных. Если для горизонтальных и вертикальных представлений это условие практически всегда может быть выполнено, то для сгруппированных и объединенных представлений, а также для сложных запросов, включающих в себя подзапросы, сопоставление данных представления и реальной таблицы не всегда может быть однозначно интерпретировано.

В соответствии с существующим стандартом SQL данные, содержащиеся в представлении, можно модифицировать только при выполнении следующих условий.

1. Модифицировать можно только данные, полученные из одной базовой таблицы данных, причем пользователь должен иметь соответствующие права доступа к этой таблице. При этом, если базовая для данного представления таблица сама по себе является представлением, то она также должна удовлетворять этому условию.

2. В корректирующем запросе не должны присутствовать данные, полученные с помощью операторов GROUP BY и HAVING.

3. В корректирующем запросе не должен использоваться оператор DISTINCT. То есть если имеются повторяющиеся строки, то они не должны исключаться из виртуальной таблицы запроса.

4. Столбцы представления должны быть простыми в том смысле, что они не должны быть вычисляемыми, полученными с помощью агрегированных функций или содержать выражения.

5. В представлении не должно содержаться вложенных запросов.

6. В представлении нельзя использовать соединение таблицы “сама с собой”.

7. Ни на один столбец в выражении оператора SELECT нельзя ссылаться более одного раза.

Соблюдение перечисленных условий необходимо прежде всего потому, что это позволяет сохранить целостность БД. При этом в ряде СУБД обеспечивается контроль не всех из этих условий, поэтому при использовании представлений необходимо контролировать осуществляемые с помощью них операции.

Похожей на представление конструкцией является курсор, поскольку при его создании используется запрос. Курсор используется только во встроенном SQL и создается следующим образом:

DECLARE <имя курсора> CURSOR
FOR <подзапрос>

Использование курсоров обусловлено необходимостью согласования включающих языков и теоретико-множественного языка SQL.

Курсор может находиться в открытом и закрытом состояниях. Открытие курсора осуществляется командой

OPEN CURSOR <имя курсора>.

При закрытии курсора используется команда

CLOSE CURSOR <имя курсора>.

При выполнении этой команды происходит выполнение запроса, описанного при определении курсора. В открытом состоянии позиция курсора может быть произвольной – перед заданной строкой, в строке, после последней строки. Однако при первом открытии курсора он устанавливается перед первой строкой. Строка, на которой установлен курсор, является текущей строкой курсора.

Курсоры бывают обновляемыми и только для чтения. Обновляемым может быть только такой курсор, который базируется только на одной таблице данных.

Контрольные вопросы

1. Что понимается под целостностью баз данных?
2. Какие виды целостности баз данных вы знаете?
3. Какие критерии лежат в основе поддержки целостности реляционной модели данных?
4. Чем различаются декларативный и процедурный механизмы обеспечения целостности?

5. Какие существуют виды декларативных ограничений целостности?

6. Что понимается под семантической целостностью?

7. В чем заключаются различия между одномоментными и отложенными ограничениями целостности?

8. Что из себя представляет механизм триггеров и когда он используется?

9. Что такое SQL? Для решения каких задач он используется? Из каких частей состоит?

10. Чем отличаются статический и динамический SQL?

11. Для чего используются языки определения данных и манипулирования данными? Назовите основные операторы этих языков. Приведите примеры их использования.

12. Приведите примеры использования оператора CREATE для создания различных объектов.

13. Приведите примеры использования оператора SELECT для выборки данных

14. Назовите операторы языка управления данными. Для каких целей они используются?

15. Какой оператор используется для задания условий поиска?

16. Какие операторы предназначены для декларативного задания ограничения целостности?

17. Какие операторы предназначены для ограничения ссылок?

18. В каких случаях применяется оператор CHECK? Приведите примеры его использования

19. Когда применяется оператор CONSTRAINT? Какие идентификаторы ограничений используются в совокупности с этим оператором? Приведите примеры.

20. Перечислите основные типы данных, которые поддерживаются в SQL?

21. Какие встроенные функции содержит в себе SQL?

22. Что представляет из себя запрос на выборку?

23. Приведите примеры запросов, результатом которых являются прямое произведение двух отношений и проекция произведения на часть столбцов двух таблиц?

24. Какие функции агрегирования используются в запросах? Приведите примеры.

25. Какие операнды используются для указания списка таблиц, из которых необходимо выбрать данные, и списка столбцов таблиц? Приведите примеры.

26. Какие операнды используются для упорядочения, сортировки данных, получаемых в результате выполнения запроса? Приведите примеры.

27. Какие типы соединения таблиц используется в запросах? Приведите примеры.

28. Чем отличаются представления от запросов на выборку? Какие операторы предназначены для создания представлений?

29. Приведите примеры горизонтального, вертикального и сгруппированного представлений.

30. Для каких целей используются сгруппированные представления? Приведите примеры.

31. При выполнении каких условий данные, содержащиеся в представлении, можно модифицировать?

32. Что из себя представляет курсор? Что общего и в чем различия между курсором и представлением?

5. ФУНКЦИОНИРОВАНИЕ И АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

5.1. Распределенная обработка данных

В современных условиях работа пользователя с небольшой по объему БД, расположенной на одном компьютере, в монопольном режиме является нехарактерной. Компьютеры объединяются в сеть (совокупность компьютеров, связанных каналами передачи данных), и возникает задача распределения приложений, работающих с одной и той же БД.

Классификация сетей может быть проведена по большому количеству их свойств, однако наиболее часто в зависимости от удаленности компьютеров друг от друга сети разделяют на локальные и глобальные (рис. 5.1).

Рис. 5.1. Классификация сетей

Под локальными обычно понимают такие сети, в которых компьютеры удалены друг от друга не более чем на несколько километров (чаще всего в одном или нескольких соседних помещениях)

и предназначены, главным образом, для обеспечения деятельности какой-то организации. Поэтому такие сети часто называют корпоративными. Локальные сети обычно предназначены для более эффективной работы с данными в совместном режиме нескольким группам пользователей – они обеспечивают доступ с одного компьютера к информации, расположенной на нескольких компьютерах.

Среди глобальных сетей различаются следующие типы: городские, региональные, национальные и транснациональные. Глобальные сети могут объединять в себе локальные сети и включать в качестве составных элементов другие глобальные сети.

Программное обеспечение любого сетевого компьютера делится на две составляющие: первая отвечает за управление ресурсами самого компьютера (прежде всего, это операционная система), а вторая – за обмен информации с другими компьютерами.

Компонентами локальных сетей являются рабочие станции (персональные компьютеры пользователей), серверы (компьютеры, которые обеспечивают взаимодействие составных частей сети и распределяют между рабочими станциями сетевые ресурсы) и кабельные линии связи. Также дополнительными компонентами являются источники бесперебойного питания, модемы, коннекторы и т. д.

В ряде случаев при наличии нескольких серверов каждый сервер обеспечивает совместную работу в сети некоторой части рабочих станций. Эта группа рабочих станций плюс управляющий сервер образуют домен.

Двумя основными двумя механизмами управления локальными сетями являются централизованный и децентрализованный. В первом случае один (или небольшое количество) компьютеров являются серверами, а остальные рабочими станциями. Во втором случае сервер фактически отсутствует, а функции управления сетевыми ресурсами распределены между рабочими станциями и могут периодически перераспределяться между ними. Основное достоинство централизованного управления заключается в высокой степени их защиты и удобство администрирования. С другой стороны, при выходе сервера из строя работа всей сети практически парали-

зуется. Именно по этой причине и используются методы децентрализованного управления.

Рассмотрим основные особенности функционирования БД в распределенных системах.

Если физически БД расположена на одном компьютере, а работу с ней в параллельном режиме осуществляет группа пользователей, каждый из которых территориально расположен на своем рабочем месте, то происходит распределенный доступ к централизованной БД. Система, реализующая такой способ, называется системой распределенной обработки данных.

Если при этом сама БД распределена по нескольким компьютерам, то такая БД называется распределенной, а система, обеспечивающая взаимодействие с ней пользователей, называется системой распределенной базы данных.

В дальнейшем обе эти системы будем называть распределенными системами.

Наиболее широко распространенным способом построения распределенных систем является архитектура клиент-сервер. Эта архитектура предполагает такой способ управления данными, при котором имеется ряд сетевых компонентов (узлов), между которыми распределяются отдельные части СУБД, каждая со своим функциональным предназначением.

Клиентом называется компьютер с размещенной на нем программой, которая обеспечивает интерфейс пользователя – получает от него запрос к данным, отсылает их серверу, получает обратно результат запроса и представляет его пользователю.

Сервер является основной структурой, обеспечивающей функции управления данными. На сервере физически хранятся программы обработки данных, которые называются хранимыми процедурами. Кроме того, на сервере могут располагаться запросы, которые называются хранимыми командами. Также на сервере может храниться и сама БД (или ее часть).

В зависимости от количества узлов сети выделяют двухзвенную и трехзвенную архитектуру модели клиент-сервер.

В двухзвенной архитектуре одним из узлов является компьютер-сервер. На этом компьютере обязательно имеется функция СУБД, отвечающая за управление данными. Вторым узлом являет-

ся компьютер-клиент. Роль его заключается в представлении данных пользователю. Моделей распределения функциональных обязанностей между сервером и клиентом может быть несколько. В одних случаях вся работа по управлению данными производится на мощном сервере, в других этим занимается клиент, а сервер выполняет лишь обработку поступивших запросов. Перечислим особенности наиболее распространенных моделей.

1. Модель удаленного доступа к данным. В этой модели на клиенте сосредоточены как функции представления данных, так и функции обработки этих данных. Прикладные программы, предназначенные для манипулирования данными, расположены именно на клиенте. Это является одним из основных недостатков данной модели, поскольку при таком подходе по сетям передается большое количество данных от сервера к клиенту, что приводит к их перегрузке и, как следствие, к потере в быстродействии всей системы в целом.

2. Модель сервера БД. В этом случае функции представления данных остаются за клиентом, в то время функции манипулирования данными передаются серверу, что приводит с существенному уменьшению нагрузки на сети.

3. Модель распределенного представления. В данной модели сервер становится еще более мощным и на нем сосредоточиваются практически все функции управления данными, а роль клиента заключается только в визуальном отображении результатов работы.

4. Модель распределенной функции. Здесь большая часть прикладных программ работы с данными реализуется на сервере. На клиенте реализуются некоторые специфичные функции обработки информации.

5. Модель распределенной БД. В этой модели на мощном клиенте располагаются не только прикладные программы, но и часть самих данных.

В трехзвенной архитектуре добавляется третий узел, называемый сервером приложений. В его функции входит обеспечение взаимодействия клиента и сервера БД. Серверов приложений может быть несколько, на них размещаются программы-приложения, которые обрабатывают поступающие от клиента запросы. В функ-

ции клиента входит представление информации пользователю посредством определенного интерфейса.

В данной архитектуре возможны более сложные структуры, например, когда сервер приложений может выступать в роли клиента для другого сервера приложений.

В распределенных системах информация, содержащаяся в БД, может храниться двумя способами – целиком на каком-либо узле системы (централизованно) или быть распределенной по нескольким узлам (децентрализованно). Во втором случае в процессе функционирования системы возникает проблема обеспечения контроля за всеми вносимыми (например, в результате транзакций) в БД изменениями и, как следствие, проблема идентичности информации, представляемой различным пользователям при их одновременной работе с данными.

Существуют две наиболее широко распространенные модели (технологии) управления данными при децентрализованном их хранении – модель распределенных БД и модель тиражирования (репликации).

В модели распределенных БД имеется так называемый глобальный словарь данных, в котором содержится информация о физическом местоположении каждой из частей распределенной БД. Доступ к данным осуществляется посредством протокола двухфазной фиксации транзакций. Суть его заключается в том, что на первой фазе выполнения транзакции над данными, находящимися на определенном узле, сообщение об этом с помощью глобального словаря данных посылается специальной программе-приложению, которая начинает контролировать ход выполнения транзакции. После завершения транзакции и получения подтверждения о ее успешном завершении (вторая фаза) приложение посылает на все узлы системы команду о фиксации внесенных в БД изменений. Таким образом, на всех узлах системы пользователи работают с данными с учетом всех последних внесенных изменений.

В модели тиражирования данных обеспечивается механизм создания копий данных на всех узлах системы. Для этого в системе имеется специальный компонент, называемый репликатором, функциональное назначение которого заключается в обеспечении идентичности всех копий.

По сравнению с моделью распределенной БД модель тиражирования при прочих равных условиях обеспечивает более высокую скорость доступа к данным, поскольку в этом случае данные всегда имеются на любом из узлов системы.

Доступ к данным различными группами пользователей всегда осуществляется либо в монопольном, либо в коллективном режимах.

Монопольный доступ используется тогда, когда по ряду причин необходимо исключить доступ других пользователей к информации. Другим случаем является такая работа с данными, когда наличие других пользователей может нарушить целостность БД (в частности, такая ситуация может возникнуть при необходимости внесений структурных изменение в БД). Для реализации монопольного доступа используется механизм блокировок данных. Этот механизм и основные особенности коллективного режима доступа к данным будут рассмотрены в следующем разделе.

5.2. Модели транзакций

Под транзакциями понимаются действия, производимые над базой данных и переводящие ее из одного согласованного состояния в другое согласованное состояние. При разработке модели транзакции прежде всего следует обеспечить поддержку целостности реляционной модели данных. Контроль целостности должен осуществляться при манипулировании данными, которое заключается в добавлении новых записей, изменении содержания имеющихся записей и удалении записей.

В настоящее время существует ряд моделей транзакций, к основным из которых относятся плоские или классические транзакции, цепочечные (многозвенные) и вложенные транзакции. Однако ко всем моделям транзакций предъявляется общий набор требований, получивший название ACID (Atomicity – атомарность, Consistency – согласованность, Isolation – изолированность, Durability – долговечность).

Атомарность – транзакция должна быть выполнена полностью или не выполнена вообще. Это означает, что либо выполняются все действия и тогда транзакция фиксируется и БД переходит в новое

согласованное состояние, либо, при невозможности выполнения всех действий транзакция откатывается назад а БД остается в исходном состоянии.

Согласованность – при выполнении транзакции БД переходит из одного согласованного состояния в другое согласованное состояние.

Изолированность – при одновременном выполнении двух или более транзакций над одним и тем же набором данных эти транзакции физически обрабатываются последовательно (изолированно друг от друга), несмотря на то что для пользователей это может выглядеть так, как будто они выполняются параллельно. По мере выполнения транзакции некоторое время данные могут находиться в несогласованном состоянии. Такие данные не должны быть видны другим транзакциям до тех пор пока все изменения не будут завершены и БД не перейдет в новое согласованное состояние (или не будет выполнен откат).

Долговечность – если транзакция выполнена успешно и выполнена ее фиксация, то все изменения в БД не могут быть потеряны ни при каких обстоятельствах.

Любая транзакция должна завершиться одним из двух возможных способов. В случае, если она завершается успешно, должна быть произведена фиксация транзакции – все результаты этой транзакции, т. е. те изменения, которые были внесены в БД, должны стать постоянными, а сама БД должна перейти в новое согласованное состояние. Если же в ходе выполнения транзакции возникает ситуация, когда транзакция не может быть успешно завершена, должен произойти так называемый откат БД в исходное состояние.

В SQL имеется два оператора, служащие для реализации одного из этих двух возможных способов завершения транзакции. Оператор COMMIT означает успешное завершение транзакции, оператор ROLLBACK отменяет все изменения, совершенные в ходе выполнения транзакции, и возвращает БД в исходное состояние.

Как уже отмечалось, существует несколько моделей транзакций. Рассмотрим кратко их отличительные особенности.

В модели плоских транзакций при выполнении транзакции либо должны успешно завершиться все производимые ей над БД действия, либо не должно выполниться ни одно из них. Если происходит

сбой при выполнении хотя бы одного действия, то все остальные действия должны быть аннулированы.

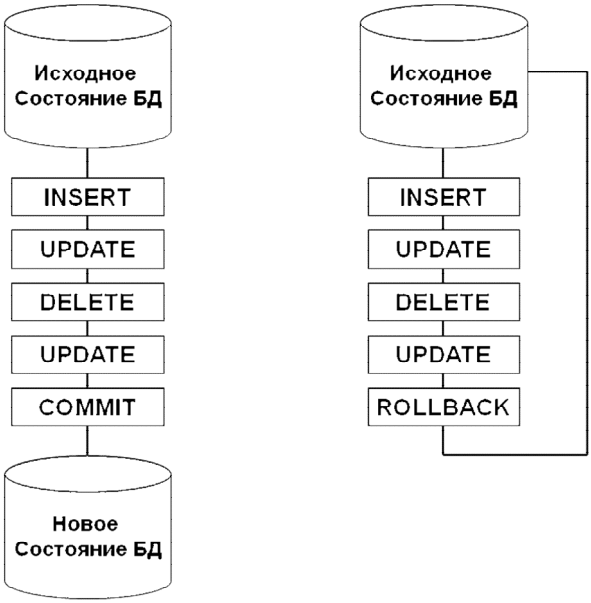


Рис. 5.2. Способы завершения плоской транзакции

На рис. 5.2 приведены возможные варианты выполнения плоской транзакции.

Таким образом, при повторном выполнении этой же самой транзакции придется выполнять все действия заново, а это значительно повышает требования к вычислительным ресурсам, снижает пропускную способность системы в целом и увеличивает время работы конкретного пользователя с БД. Особенно это важно для крупных организаций, с БД которых работает одновременно большое количество пользователей.

Для устранения этого недостатка был разработан механизм контрольных точек, который позволил уменьшить количество повторно выполняемых действий при неудачном выполнении транзакции и последующем ее повторном выполнении. Контрольные точки устанавливаются в прикладной программе, которая осуществляет транзакцию и их назначение заключается в том, чтобы отмечать

моменты, начиная с которых возможно продолжение выполнения транзакции при возникновении проблем.

Происходит это следующим образом (рис. 5.3). При достижении контрольной точки в транзакции создается новое действие, которое запускается на выполнение. Последнее из этих действий, соответствующее последней контрольной точке, фиксирует всю транзакцию. При возникновении проблем на каком-то этапе операции, выполненные до текущей контрольной точки не отменяются, и поэтому при повторном выполнении транзакции их повторять не придется. Другими словами, откат происходит не в начало транзакции, а до определенной контрольной точки.

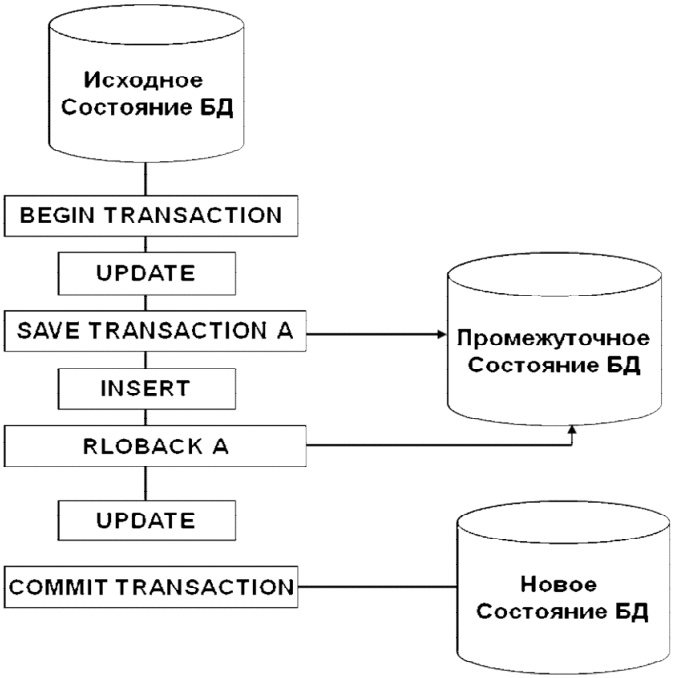


Рис. 5.3. Реализация механизма контрольных точек

В модели вложенных транзакций имеется так называемая головная транзакция, которая управляет всей последовательностью

действий, которые представляют из себя транзакции разной степени вложенности.

Одной из наиболее важных задач при разработке механизмов выполнения транзакций является обеспечение возможности восстановления согласованного состояния данных при различных сбоях. Восстановление данных необходимо производить в следующих ситуациях (рис. 5.4):

- индивидуальный откат транзакции;
- потеря содержимого оперативной памяти (мягкий сбой);
- выход из строя носителя внешней памяти (жесткий сбой).

Рис. 5.4. Случаи необходимости восстановления данных

Общий принцип механизма восстановления данных можно сформулировать следующим образом: в восстановленном состоя-

нии данные должны содержать результаты всех зафиксированных транзакций и не содержать результаты незафиксированных транзакций.

Для реализации возможности восстановления данных существует специальная структура, которая называется журналом транзакций. Основным принципом журнала транзакции является обеспечение протокола, называемого Write Ahead Log (WAL) – “пиши сначала в журнал”. Это означает, что любая запись об изменении объекта данных должна сначала попадать во внешнюю память журнала транзакций, и только потом во внешнюю память базы данных. Таким образом, если во внешней памяти базы данных содержится информация о некотором объекте, над которым уже выполнена операция модификации, то в журнале транзакций заведомо имеется соответствующая запись.

Иногда для обеспечения возможности восстановления базы данных одного журнала транзакций бывает недостаточно: в частности, при жестком сбое возможна потеря содержимого не только базы данных, но и самого журнала. В этом случае основой для восстановления данных является создание архивных копий базы данных, которые могут храниться на различных серверах.

Главная сложность при выполнении транзакций возникает при параллельном выполнении двух или более транзакций над одним и тем же объектом БД. Основными проблемами, возникающими при этом, являются проблемы пропавших изменений, промежуточных и несогласованных данных, строк-призраков. Более подробно об этих проблемах, проиллюстрированных разнообразными примерами, можно прочитать в приводимой в конце настоящего пособия литературе. Все они сводятся к тому, что одна из транзакций получает доступ к объекту БД, изменяемому в данный момент другой транзакцией, т. е. видит БД в несогласованном состоянии.

Для избежания подобных проблем разработана специальная процедура согласованного выполнения параллельных транзакций, которая называется сериализацией транзакций. Эта процедура удовлетворяет следующим критериям.

1. В ходе выполнения транзакции пользователь должен видеть только согласованные данные.

2. Гарантированно поддерживается принцип независимого выполнения параллельных транзакций. Это означает, что результат выполнения первой транзакции будет таким же, как если бы вначале выполнялась эта транзакция, а уже затем вторая, или наоборот, сначала вторая, а потом первая.

Такая процедура всегда гарантирует, что каждый пользователь при работе с данными работает с ними так, как будто не существует других пользователей, одновременно работающих с теми же данными.

Для обеспечения возможности параллельного выполнения транзакций строится специальный сериальный план, который обеспечивает выполнение того, что результат параллельного выполнения транзакций эквивалентен результату некоего последовательного выполнения этих же транзакций.

Для реализации такого плана разработан специальный механизм, получивший название механизма блокировок транзакций или синхронизационных захватов. Блокировка запрещает некоторые операции над данными в том случае, если эти данные обрабатываются другим пользователем. Существует два типа блокировок – разделяемая или совместная блокировка S (Shared) и эксклюзивная или монопольная блокировка X (eXclusive). В режиме разделяемой блокировки один и тот же объект данных может быть доступен сразу нескольким транзакциям, но только в режиме чтения. В режиме эксклюзивной блокировки объект доступен единственной транзакции, а все остальные транзакции находятся в режиме ожидания.

Сериализуемость транзакций будет всегда гарантирована, если блокировки удовлетворяют следующему условию: ни одна транзакция не может установить свою блокировку на объект до тех пор, пока с этого объекта не будет снята уже установленная блокировка другой транзакцией.

Имеются следующие уровни блокировок: уровень всей БД, уровень совокупности связанных таблиц, одной таблицы, совокупности связанных записей, одной записи, поля.

Несмотря на то, что использование блокировок позволяет решить ряд проблем при параллельном выполнении транзакций, этот метод имеет один существенный недостаток, который называется

проблемой тупиков или взаимной блокировкой транзакций. Эту проблему легче всего понять на следующем примере.

На рис. 5.5 изображена ситуация, когда над двумя объектами БД в параллельном режиме совершают работу две транзакции. Вначале транзакция 1 захватывает объект 1 БД в нежестком режиме, а чуть позже транзакция 2 в жестком режиме захватывает объект 2. Чуть позже транзакция 1 принимает решение произвести модификацию объекта 1, для чего захватывает его в жестком режиме.

Закончив работу с объектом 2, вторая транзакция для дальнейшей работы пытается в жестком режиме захватить объект 1, однако сделать этого не может, поскольку он уже в жестком режиме захвачен первой транзакцией. Поэтому вторая транзакция входит в режим ожидания завершения работы над этим объектом первой транзакции. Однако первая транзакция для завершения работы с первым объектом должна произвести модификацию второго объекта и, пытаясь захватить его в жестком режиме, ждет завершения работы второй транзакции.

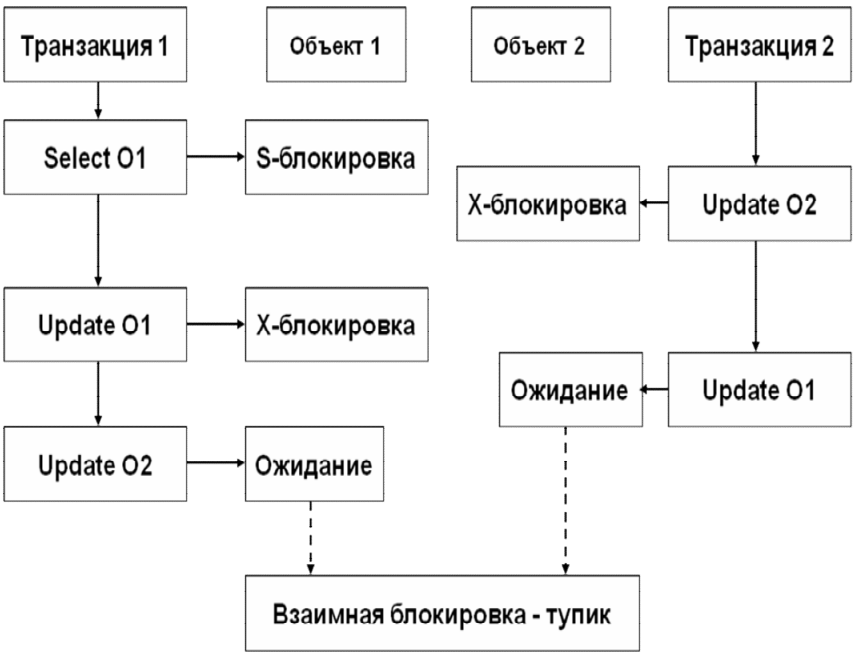


Рис. 5.5. Взаимная блокировка транзакций

Таким образом, каждая транзакция ждет завершения работы другой, и это ожидание будет длиться бесконечно долго, что и называется тупиком.

Для разрешения тупиковых ситуаций разработан ряд схем, и данная область интенсивно развивается в настоящее время. В современных СУБД процесс обнаружения тупиков автоматизирован. В его основе лежит построение графа ожидания транзакций, пример которого приведен на рис. 5.6.

На этом рисунке каждый круг соответствует транзакции с определенным номером. Если одна транзакция для дальнейшего продолжения своей работы ждет завершения работы другой транзакции, то от нее в направлении этой транзакции направлена стрелка.

На рисунке имеется циклическая конструкция (обведена), наличие которой, как нетрудно видеть, соответствует наличию тупика. Разрыв любой стрелки внутри этой конструкции (т. е. отмена любой из транзакций под номерами 5, 6, 7 и 8) автоматически приведет к устранению тупика и продолжению работы всех оставшихся транзакций.

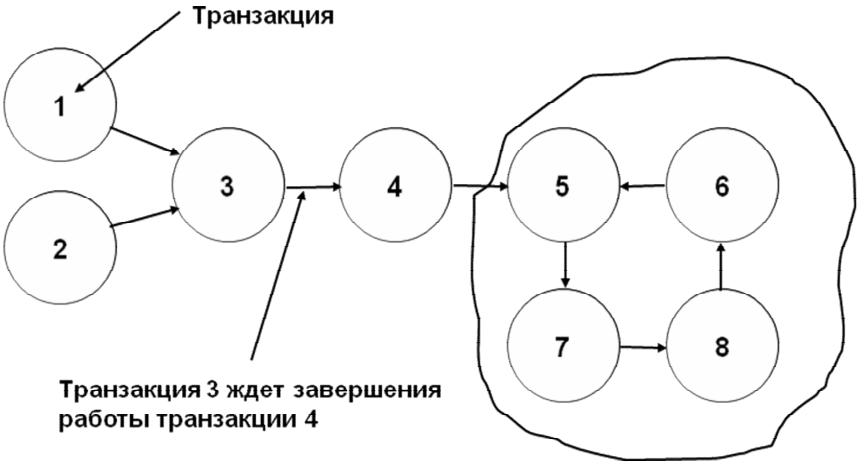


Рис. 5.6. Граф ожидания транзакций

В самом общем случае разрешение проблемы тупика начинается с поиска так называемой транзакции-жертвы, т. е. с такой тран-

закции, которую можно отменить, совершив откат БД в исходное состояние, обеспечив таким образом остальным транзакциям продолжить свою работу. Самым сложным при этом является выбор критерия, в соответствии с которым выбирается транзакция-жертва. Для этого вводится понятие стоимости транзакции, которая определяется исходя из ряда факторов – время выполнения транзакции, количество выполненных операций, приоритет транзакции и т. д.

Тупиковые ситуации возникают гораздо реже, если система имеет возможность дифференцируемого наложения блокировок на объекты данных различного типа. В этом случае вводится уровень блокировки.

В современных системах требуемый уровень блокировки определяется исходя из того, какая именно операция выполняется. Так, например, при выполнении операции удаления отношения блокировка устанавливается целиком на это отношение, а при удалении кортежа – только на этот кортеж. Для этого вводится специальный протокол, называемый протоколом гранулированных захватов и при этом вводятся новые типы блокировок:

IS (Intented for Shared Lock) – по отношению к некоторому составному объекту данный тип блокировки означает намерение захватить часть этого объекта в совместном режиме. Если, например, необходимо прочитать часть кортежей из данного отношения, то это отношение вначале захватывается в режиме IS.

IX (Intented for eXclusive Lock) - по отношению к некоторому составному объекту данный тип блокировки означает намерение захватить часть этого объекта в монопольном режиме. При необходимости удаления части кортежей из данного отношения это отношение вначале захватывается в режиме IX.

SIX (Shared, Intented for eXclusive Lock) - по отношению к некоторому составному объекту данный тип блокировки означает совместный захват этого объекта с намерением впоследствии захватывать части этого объекта в монопольном режиме. Этот тип блокировки используется, если, например, выполняется просмотр записей отношения и при необходимости некоторые записи нужно удалить.

Для реализации протокола гранулированных захватов необходимо выполнение следующего условия: прежде чем устанавливается S-блокировка (или X-блокировка) на часть некоторого объекта (например, на кортеж отношения), должна быть установлена IS-блокировка (или IX-блокировка) на весь объект (например, на все отношение). Здесь вместо IS- и IX-блокировок могут быть более сильные блокировки.

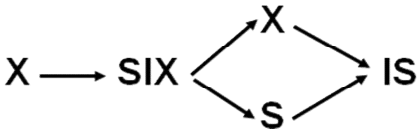


Рис. 5.7. Диаграмма приоритетов блокировок

На рис. 5.7 приведена диаграмма приоритетов блокировок.

Табл. 5.1 представляет из себя так называемую матрицу совместимости блокировок. Плюс означает совместимость группировок, минус – несовместимость.

Таблица 5.1. Матрица совместимости группировок

Тип блокировки	X	S	IX	IS	SIX
Отсутствует	+	+	+	+	+
X	-	-	-	-	-
S	-	+	-	+	-
IX	-	-	+	+	-
IS	-	+	+	+	+
SIX	-	-	-	+	-

Еще одним понятием, относящимся к проблеме параллельного выполнения транзакций, является уровень изолированности пользователей. Зачастую скорость доступа к данным является существенно более важным показателем по сравнению с точностью отображения самих данных. В такой ситуации уровни блокировок транзакций могут быть частично смягчены. Существуют четыре уровня изолированности.

Уровень serializable является самым жестким и обеспечивает полную изолированность транзакций. Этот уровень обеспечивает максимальную степень целостности данных и как правило устанавливается по умолчанию. На данном уровне изолированности каждая транзакция выполняется изолированно.

Уровень repeatable read (подтвержденное чтение) отказывает транзакции в доступе к промежуточным или окончательным результатам других транзакций. При этом из всех проблем остается только проблема строк-призраков. Данный уровень допускает вставку новой записи в БД, обрабатываемую другой транзакцией.

На уровне read commited (чтение с фиксацией) транзакция не имеет доступа к промежуточным результатам других транзакций. Окончательные результаты других транзакций на этом уровне доступны, поэтому проблемы строк-призраков и несогласованных данных могут возникнуть. На этом уровне допускается выполнение запроса, однако лишь при условии, что результаты параллельных транзакций были зафиксированы.

Самый низкий уровень изолированности называется read uncommitted (грязное чтение или чтение без фиксации). На этом уровне предотвращается только проблема пропавших обновлений и допускается выполнение запроса вне зависимости от того, были зафиксированы результаты параллельных транзакций или нет.

В табл. 5.2 показано, какие проблемы разрешаются на каждом уровне изолированности.

Таблица 5.2. Уровни изолированности пользователей

Уровень изолированности	Проблема			
	Пропавшие обновления	Промежуточные данные	Несогласованные данные	Строки-фантомы
Serializable	Не возникает	Не возникает	Не возникает	Не возникает
Repeatable read	Не возникает	Не возникает	Не возникает	Сохраняется
Read commited	Не возникает	Не возникает	Сохраняется	Сохраняется
Read uncommited	Не возникает	Сохраняется	Сохраняется	Сохраняется

5.3. Защита информации в базах данных

Целью защиты информации является обеспечение безопасности ее хранения и обрабатывания. Процесс построения эффективной защиты начинается на начальных стадиях разработки программного обеспечения и включает в себя выявление наиболее уязвимых элементов системы, определение степени угрозы для составных элементов системы, что приводит к формированию критериев относительно требований к системе защиты и выбору методов и средств для их реализации.

Существуют два наиболее общих подхода к вопросу обеспечения безопасности данных: избирательный и обязательный (рис. 5.8). Эти два подхода обладают следующими свойствами.

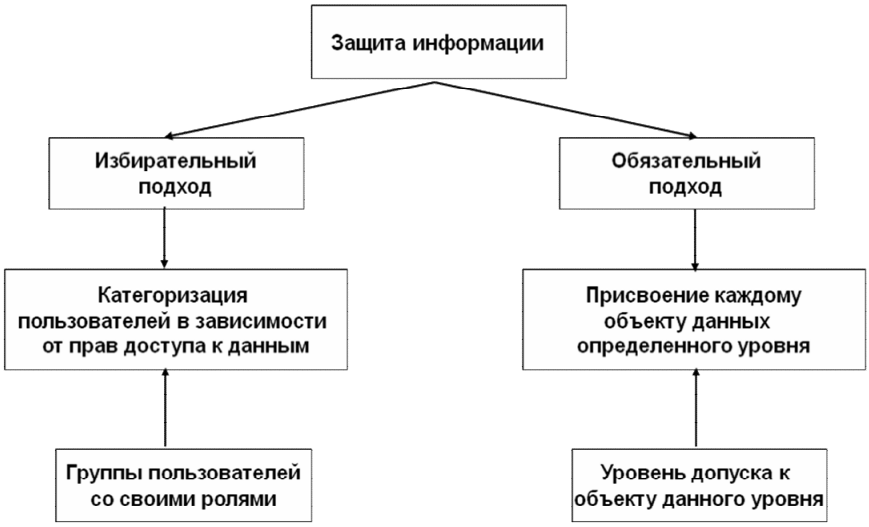


Рис. 5.8. Способы защиты информации

При избирательном подходе пользователи обладают различными правами (привилегиями или полномочиями) при работе с данными. Разные пользователи могут обладать разными правами доступа к одному и тому же объекту данных. Избирательные права характеризуются значительной гибкостью.

Для реализации избирательного принципа предусмотрены следующие методы. В БД вводится новый тип объекта – пользователи. Каждому пользователю присваивается уникальный идентификатор. Для дополнительной защиты каждый пользователь получает уникальный пароль, причем если идентификаторы пользователей доступны системному администратору, то пароли, как правило, известны только самим пользователям.

В случае обязательного подхода каждому объекту данных присваивается некоторый уровень, и каждый пользователь обладает некоторым уровнем допуска к данному объекту. При таком подходе доступом к определенному объекту данных обладают только пользователи с соответствующим уровнем допуска.

Пользователи могут быть объединены в специальные группы. Один пользователь может входить в несколько групп. Группой, для которой определен минимальный стандартный набор прав, является группа PUBLIC. По умолчанию предполагается, что каждый новый пользователь относится к группе PUBLIC.

Привилегии или полномочия пользователей или групп пользователей – это набор действий (операций), которые они могут выполнять над объектами БД.

В последних версиях ряда СУБД появилось понятие «роль». Ролью называется поименованный набор полномочий пользователей. Существует ряд стандартных ролей, имеются возможности создания новых ролей с различными полномочиями. Введение ролей позволяет упростить управление привилегиями пользователей и структурировать этот процесс. Кроме того, введение ролей не связано с конкретными пользователями, поэтому роли могут быть определены и сконфигурированы до того, как определены пользователи системы. Пользователю или группе пользователей может быть назначена одна или несколько ролей.

В самом общем случае концепция обеспечения безопасности БД заключается в поддержании двух принципов – проверки полномочий и проверки подлинности.

Проверка полномочий заключается в том, что каждому пользователю соответствует набор действий, которые он может осуществлять над каким-либо объектом БД. Как правило, полномочия пользователей хранятся в специальных системных таблицах СУБД,

которая осуществляет их проверку при выполнении каждой операции. Проверка полномочий приобретает в настоящее время все большую важность, что связано прежде всего с массовым распространением распределенных вычислений.

Проверка подлинности означает, что должен быть достоверно подтвержден факт того, что пользователь действительно тот, за которого он себя выдает (аутентификация пользователя). Для обеспечения безопасности в СУБД должна присутствовать модель проверки подлинности, которая будет выполнять подтверждение заявленных пользователями идентификаторов.

Для реализации полномочий в современных СУБД строится система назначений полномочий, которая имеет иерархический характер (рис. 5.9).

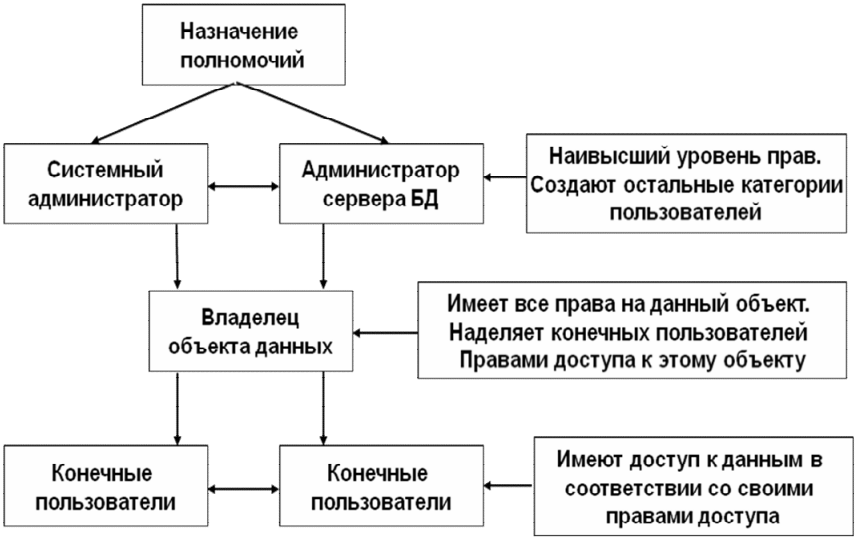


Рис. 5.9. Система назначения полномочий

Самыми высокими правами и полномочиями обладает системный администратор или администратор сервера БД. Именно эта категория пользователей имеет право создавать другие категории пользователей и наделять их определенными полномочиями. Эти полномочия строятся по следующему принципу. Каждый объект БД имеет своего владельца – как правило, пользователя, который этот объект создал. Владелец объекта обладает всеми правами на

этот объект, включая право предоставлять другим пользователям различные полномочия для работы с данным объектом (или забирать эти полномочия).

Рассмотренные вопросы обеспечения безопасности не решают всех проблем защиты информации. В современной концепции защиты информации считается, что безопасность информации нарушается вследствие реализации одной или нескольких угроз, что означает выполнение преднамеренного или случайного действия, которое может привести к сбою системы защиты.

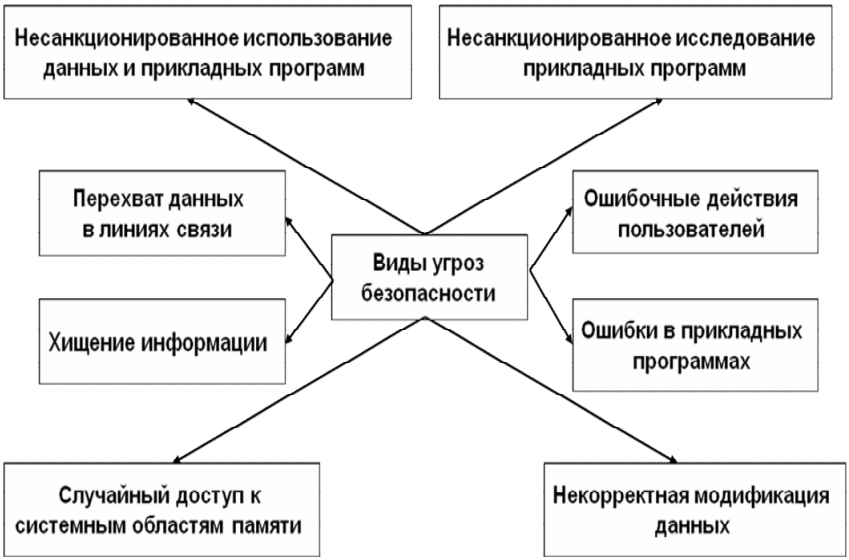


Рис. 5.10. Классификация видов угроз

Основными видами угроз являются следующие (рис. 5.10):

несанкционированное использование данных и прикладных программ (копирование, модификация, удаление), а также исследование таких программ для последующего вторжения в систему;

- случайный доступ к системным областям памяти;
- некорректная модификация данных;
- ошибочные действия пользователей;
- ошибки в прикладных программах;
- перехват данных в линиях связи;

- хищение информации (дисков или иных носителей данных).

В результате нарушения безопасности возможны следующие последствия:

- потеря секретных сведений;
- сбой системы – замедление ее работы или полная остановка;
- потеря содержимого внешней памяти (жесткий сбой);
- материальный ущерб.

Исходя из этого, имеются три основные задачи защиты: защита от хищения, от потери и от сбоев и отказов системы.

Под понятием защиты от хищения подразумевается предотвращение физического хищения, несанкционированного доступа к информации и несанкционированного копирования (с целью дальнейшего размножения) данных и прикладных программ.

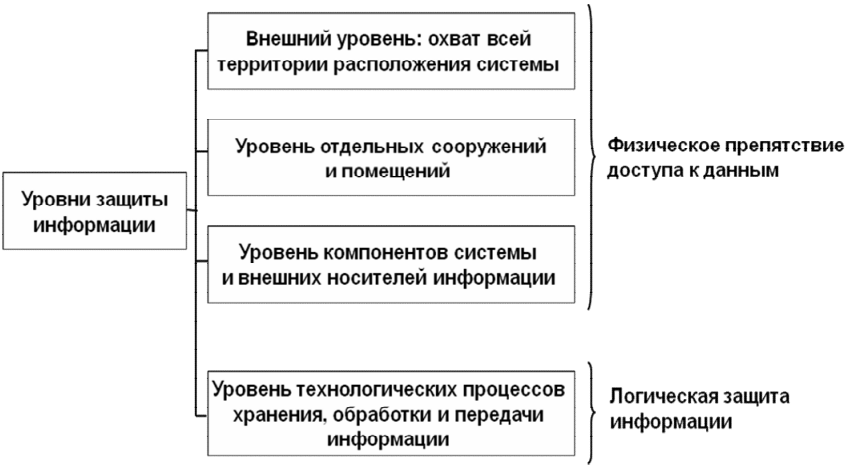


Рис. 5.11. Уровни защиты информации

Для защиты от потери необходим механизм поддержания целостности (физической, логической и семантической) и корректности хранимой в БД информации.

Степень защиты от сбоев и отказов системы определяется надежностью системных компонентов – процессора, внешней памяти, других устройств, а также операционной системы и ряда других

программ. Под надежностью понимается способность системного компонента однозначно и своевременно выполнить возложенные на него функции.

Для организации защиты информации предусматривается четыре защитных уровня (рис. 5.11) – внешний уровень, охватывающий всю территорию расположения системы, уровень отдельных сооружений и помещений, уровень компонентов системы и внешних носителей информации, уровень технологических процессов хранения, обработки и передачи информации. Первые три уровня обеспечивают физическое препятствие доступа к данным, а последний уровень предусматривает логическую защиту информации в том случае, когда физический доступ к ней уже имеется.

Рис. 5.12. Методы защиты информации

Существующие методы защиты делятся на четыре основных класса – физические, аппаратные, программные и организационные (рис. 5.12).

Физическая защита используется, главным образом, на верхних уровнях и состоит в физическом преграждении доступа посторон-

них лиц. Для реализации физической защиты используются стандартные средства, основанные на современных технологиях: детекторы движения объектов, телевизионные системы наблюдения за территорией объекта, сенсорные системы, средства защиты от наблюдения и прослушивания, системы нейтрализации излучений и т. д.

Аппаратная защита реализуется аппаратурой в системе ЭВМ или с помощью специализированных устройств. Основными аппаратными средствами защиты являются системы защиты процессоров, внешней памяти, устройств ввода-вывода, системы передачи информации по линиям связи, системы электропитания и т .д.

Программная защита обеспечивается посредством операционных систем, программ обслуживания, антивирусных пакетов и т. д.

Для обеспечения программно-аппаратной защиты имеются следующие способы.

1. Защита от несанкционированного доступа к данным со стороны пользователей и прикладных программ. Вопросы реализации этого способа рассмотрены выше.

2. Защита от несанкционированного использования данных при наличии доступа к ним. Для реализации этого способа в СУБД должны быть предусмотрены средства регистрации запросов к данным со стороны пользователей и прикладных программ, а также средства сигнализации в случаях попыток несанкционированного использования этих данных. При этом используются механизмы защиты от различных вариантов несанкционированного использования – от просмотра, копирования, модификации, удаления данных, исследования программ. Под исследованием программ понимается попытка изучения системы защиты этой программы

3. Защита от некорректного использования ресурсов. Этот способ предусматривает, прежде всего, изоляцию участков оперативной памяти, которые выделяются различным программам, защиту системных областей внешней памяти и контроль допустимости команд процессора.

4. Структурная, функциональная и информационная избыточность.

Под структурной избыточностью понимается резервирование аппаратных компонентов – дублирование серверов обработки, про-

цессоров, жестких дисков, схем устройств. Обязательным условием является обеспечение бесперебойности питания. Функциональная избыточность означает резервирование функций управления, хранения и обработки информации, то есть реализация этих функций несколькими элементами системы. Информационная избыточность означает периодическое копирование и архивирование наиболее ценной информации.

5. Обеспечение высокого качества используемых программно-аппаратных средств.

Организационная защита обеспечивается с помощью различных организационно-технических мероприятий, в том числе принятие законодательных актов, инструкций по защите информации.

5.4. Настройка и администрирование СУБД

Для успешного функционирования разработанной ИС недостаточно выбора СУБД и сервера БД. В процессе эксплуатации всегда на разных этапах приходится выполнять различные настройки и функции администрирования.

Рис. 5.13. Классификация администраторов информационных систем

Основными задачами администрирования являются защита информации и разграничение прав доступа различным категориям пользователей. К числу других задач относятся выбор способа размещения файлов на диске, определение требуемого объема дисковой памяти, распределение данных на диске, резервное копирование данных.

Для решения этих задач привлекается категория специалистов, называемых администраторами ИС и обеспечивающих ее функционирование на всех этапах жизненного цикла (рис. 5.13). В зависимости от объема и сложности ИС группы администраторов могут различаться как по количеству специалистов, так и набору выполняемых ими функций. Основными категориями администраторов ИС являются следующие.

- системные аналитики;
- проектировщики структур данных;
- проектировщики процессов обработки данных;
- системные, прикладные специалисты и операторы;
- специалисты по техническому обслуживанию.

Администраторов БД можно разделить на две категории.

1. Администраторы данных. Эта группа администраторов на начальной стадии разработки БД отвечает за оптимальную организацию ее структуры с точки зрения обеспечения одновременной работы с ней множества пользователей. На стадии эксплуатации БД эта группа отвечает за корректность работы с данными в многопользовательском режиме. На стадии развития и реорганизации основной задачей группы является обеспечение корректной реорганизации БД без прекращения ее текущей эксплуатации.

2. Администраторы приложений. Эта группа администраторов функционирует на стадиях проектирования, создания и реорганизации БД. В частности, администраторы приложений координируют работу специалистов при разработке прикладных программ.

Основные функции администраторов ИС заключаются в следующем:

1. Описание и анализ предметной области на начальном этапе проектирования БД – выявление ограничений целостности, определение статуса информации с точки зрения ее доступности, опре-

деление потребностей пользователей, определение объемно-временных характеристик обработки данных.

2. Проектирование структуры БД – определение состава и структуры файлов БД и связей между ними, выбор методов упорядочения данных и методов доступа к информации.

3. Задание ограничений целостности при описании структуры БД и процедур обработки БД – задание декларативных ограничений целостности, определение динамических ограничений целостности в процессе изменения информации, определение ограничений целостности, вызванных структурой БД, разработка процедур ограничения целостности при вводе и корректировке данных.

4. Выбор способа размещения файлов на диске. В большинстве случаев имеется два способа размещения файлов БД на дисках – на так называемых “чистых” дисках или в файловой структуре операционной системы. В первом случае управление данными производится средствами самих СУБД.

Преимущество хранения данных на “чистых” дисках заключается в том, что при этом более эффективно используется внешняя память (достигается экономия внешней памяти за счет устранения необходимости организации файловой системы), что приводит к увеличению производительности системы в целом.

Основные достоинства файловой системы:

- использование файловой системы обладает большей гибкостью, поскольку в этом случае администратор имеет стандартные средства (программы) обслуживания файлов;

- в ряде случаев ввод-вывод данных через файловую систему обеспечивает оптимизацию, что СУБД реализовать не в состоянии.

5. Определение требуемого объема дисковой памяти. Необходимо принимать во внимание, что для обработки данных СУБД использует достаточно большой объем служебной информации, размещаемой на дисках – описание схемы БД, табличные индексы, временные таблицы, выделенное пространство для хэш-таблиц и индексов, пространство для сортировки, файлы журнала, архивы и т. д. При этом достаточно часто информация об объеме служебной информации отсутствует, поэтому следует исходить из предположения, что он может превосходить объем собственно самих данных. Кроме того, для повышения безопасности данных и улучше-

ния степени их защиты файлы журналов транзакций и архивов целесообразно размещать на иных физических дисках, отличных от дисков с данными.

6. Распределение информации на дисках. Критерием здесь является обеспечение рационального и экономичного способа работы с данными. Одним из наиболее оптимальных способов является использование четырех дисков: для операционной системы, для данных, для журналов транзакций и для индексов. Это обусловлено тем, что совместное хранение разнородной информации на одних и тех же дисках приводит к потере производительности, вызванной перегрузкой системы ввода-вывода.

7. Определение формата начальной загрузки БД – разработка форм ввода данных и контроль ввода, подготовка исходных данных, создание прикладных программ.

8. Ведение БД – определение стратегии изменения, добавления и удаления данных, разработка технологии проверки соответствия вводимых данных реальному состоянию предметной области.

9. Защита данных от несанкционированного доступа, включающая регистрацию пользователей и определение их прав, определение допустимых операций над данными для конкретных пользователей, выбор средств защиты данных, шифрование данных при возникновении такой необходимости.

В рамках этих мероприятий производятся такие действия, как периодическое тестирование средств защиты данных, анализ обращений пользователей к данным, фиксация попыток несанкционированного доступа к данным, анализ случаев нарушения системы безопасности данных с возможной модификацией средств защиты для предотвращения таких случаев в дальнейшем.

10. Обеспечение защиты данных от потери. Для этой цели должна быть разработана система резервного копирования информации. Резервная копия может быть как точной копией исходной БД, так и архивной копией. Архивирование может осуществляться аппаратно или программно. Аппаратное архивирование является более предпочтительным с точки зрения временных затрат, но приводит к увеличению аппаратной части.

Резервное копирование может осуществляться непосредственно во время работы с БД (on-line) или в другое время. Копирование

может начаться по команде оператора или автоматически в заданное время.

При определении типа устройства для размещения резервной копии основным критерием является временной интервал, отводимый на процедуру копирования. Для решения вопроса о том, с какой частотой проводить резервное копирование, исходят из следующих соображений. On-line-копирование выбирается тогда, когда работа с БД происходит либо круглосуточно, либо когда в работе системы имеются окна, продолжительность которых достаточна для создания копии. В остальных случаях копирование происходит, как правило, в конце рабочего дня или рабочей недели.

11. Обеспечение возможности восстановления данных. Это достигается за счет разработки организационных средств архивирования и принципов восстановления данных, журнализации и разработки программных средств восстановления данных.

12. Анализ эффективности функционирования БД и развитие ее при необходимости – анализ показателей функционирования ИС, реструктуризация данных, изменение состава БД, развитие программных и технических средств ведения БД.

13. Работа с пользователями, включающая сбор информации об изменении предметной области, обучение пользователей, проведение консультаций с ними, анализ оценки пользователями работы ИС, определение регламента работы пользователей с данными.

14. Поддержание работоспособности и улучшение программных средств – анализ информации о новых программных продуктах, их приобретение, установка, другие действия по развитию программного обеспечения.

15. Организационно-методическая работа – выбор или создание методики проектирования БД, определение целей и стратегии развития ИС, планирование мероприятий, разработка общих словарей-справочников проекта БД и концептуальной модели, стыковка внешних моделей разрабатываемых приложений, обеспечение возможности комплексной отладки множества приложений, взаимодействующих с БД.

В связи с увеличением сложности и масштабности современных ИС, их высокая стоимость, и, как следствие, высокая стоимость затрат, связанных с ошибками, допущенными при администриро-

вании, в настоящее время большое внимание уделяется разработке средств автоматизированного администрирования ИС.

Во всех современных СУБД предусмотрены средства автоматизированного администрирования. Кроме того, к настоящему времени разработан отдельный пакет специализированного программного обеспечения, предназначенного для задач администрирования ИС – DataBase Administration (DBA). Использование этого пакета позволят решать следующий круг задач:

1. Непрерывный мониторинг работы ИС, включающий слежение за использованием ресурсов, ведение статистики, обнаружение, фиксация и исправление возникающих сбоев.

2. Наблюдение и анализ функционирования объектов БД – планирование необходимых вычислительных мощностей, определение критериев для слежения за объектами данных.

3. Оптимизация функционирования ИС, включающая оптимизацию хранения данных и работы сервера БД – анализ свободного дискового пространства, дефрагментация, анализ значений параметров быстродействия системы, перенос части данных на новое дисковое пространство, подключение новых СУБД к процессу ведения ИС.

4. Сопровождение БД, файловых систем, табличных пространств – перенос объектов на новое пространство, в другую СУБД, на другой компьютер.

Контрольные вопросы

1. Какие сети называются локальными, а какие – глобальными?
2. Какие типы глобальных сетей вы знаете?
3. Каким должно быть программное обеспечение сетевого компьютера?
4. Какие компоненты содержат в себе локальные сети?
5. Какие существуют механизмы управления локальными сетями?
6. Что называется распределенными системами?
7. Что из себя представляет глобальный словарь данных?
8. Какие имеются способы доступа к данным для различных категорий пользователей?
9. Что называется транзакцией?

10. Какие вы знаете модели транзакций?

11. Дайте определения свойствам атомарности, долговечности, изолированности и согласованности транзакции?

12. Какие операторы SQL предназначены для реализации транзакции?

13. Что называется индивидуальным откатом транзакции?

14. Какие ситуации называются мягким и жестким сбоем?

15. Что такое точка сохранения?

16. Какие виды журналов транзакций вы знаете? Для каких целей они используются?

17. Перечислите основные проблемы, которые возникают при параллельном выполнении транзакции?

18. В чем заключается сущность механизма сериализации транзакций? Каким критериям он должен соответствовать?

19. В чем заключается сущность процедуры блокировки транзакции? Какие типы блокировок вы знаете?

20. Какие уровни изолированности пользователей вы знаете? Какие проблемы предотвращаются на каждом из этих уровней?

21. В чем заключается цель защиты информации в базах данных?

22. Какие в настоящее время существуют механизмы обеспечения безопасности баз данных?

23. В чем заключается сущность принципов проверки полномочий и проверки подлинности?

24. Какие категории пользователей баз данных вы знаете? Как в общем случае распределяются между ними полномочия?

25. Назовите основные виды угроз безопасности базам данных? К каким последствиям может привести реализация этих угроз?

26. Сколько защитных уровней предусмотрено для организации защиты информации в базах данных?

27. Какие способы имеются для обеспечения программно-аппаратной защиты информации?

28. Назовите основные категории администраторов современных информационных систем. В чем заключаются функции каждой из категорий?

29. Назовите основные способы размещения файлов базы данных на дисках.

30. Что из себя представляет гранулированный захват? В каких случаях он используется?

31. Что такое матрица совместимости блокировок?

32. Какие задачи могут быть решены с использованием пакета DBA?

33. В чем заключаются различия в функциях администраторов данных и администраторов приложений?

34. Перечислите несколько критериев, в соответствии с которыми может быть определена стоимость транзакции?

СПИСОК ЛИТЕРАТУРЫ

Основная

1. Диго С.М. Базы данных. М.: Финансы, 2005.
2. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. М.: Вильямс, 2003.
3. Грофф Дж., Вайнберг П. SQL: полное руководство. Пер. с англ. – К.: BHV, 2001.
4. Кузнецов С.Д. SQL. Язык реляционных баз данных. М.: Майор, 2001.
5. Дейт К. Введение в системы баз данных. М.: Вильямс, 2000.
6. Карпова Т.С. Базы данных. Модели, разработка, реализация. СПб: Питер, 2001.
7. Кренке Д. Теория и практика построения баз данных. С.-Пб.: Питер, 2003.
8. Советов Б.Я., Цехановский В.В., Чертовской В.Д. Базы данных. Теория и практика. М.: Высшая школа, 2005.
9. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс. М.: Вильямс, 2003.
10. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных. С.-Пб.: Корона-принт, 2004.

Дополнительная

1. Мартин Дж. Организация баз данных в вычислительных системах. М.:1999.
2. Дунаев С. Б. Доступ к базам данных и техника работы в сети. Практические приемы современного программирования. М.: Диалог-МИФИ, 1999.
3. Арсеньев Б.П., Яковлев С.А. Интеграция распределенных баз данных. С-Пб.: Лань, 2001.
4. Полякова Л.Н. Основы SQL. Курс лекций. М.: ИНТУИТ.РУ, 2004.
5. Харрингтон Джен Л. Проектирование реляционных баз данных. М.: Лори, 2006.

6. Ролланд Ф.Д. Основные концепции баз данных. М.: Вильямс, 2002.

7. Кузнецов С.Д. Основы баз данных. Курс лекций. М.: ИНТУИТ.РУ, 2005.

8. Вендров А.М. проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2004.

ПРИЛОЖЕНИЕ. ПРИМЕР РАЗРАБОТКИ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ В СУБД DELPHI 7.0

Приводимый ниже пример описывает процесс создания и последующего управления реляционной базы данных в одной из наиболее популярных на сегодняшний день среде объектно-ориентированного визуального программирования Delphi 7. Основное назначение данного примера заключается в передаче студентам основных практических навыков реализации идей и методов, используемых в современных реляционных системах управления БД.

Повышенное внимание акцентируется на анализе предметной области, исходя из постановки задачи, разработке схем отношений на основе принципов нормализации. Исследуются возможности создания запросов к БД и отчетов, фильтрации данных и кэширования изменений. Отдельно рассматриваются вопросы о выполнении операций транзакций над БД – создания, удаления и изменения данных из взаимосвязанных таблиц.

Анализ и описание предметной области БД

База данных проектируется для предприятия, занимающегося разработкой информационных программных продуктов. С целью оптимизации деятельности предприятия его структура предусматривает ряд отделов, на каждый из которых возлагается определенный набор функций, связанных с разработкой конкретной части программного продукта. Имеются отделы общего назначения – бухгалтерия, отдел кадров, хозяйственный и административный отделы. Кроме того, имеются отделы, деятельность которых связана с анализом внешнего рынка, маркетинговой политикой предприятия, рекламой. Каждый сотрудник закреплен за определенным отделом.

Разрабатываемый предприятием программный продукт в виде файлов размещается на нескольких серверах. С целью обеспечения возможности доступа сотрудникам различных отделов с различными правами к одним и тем же файлам имена файлов на различных серверах могут совпадать, а имена одного файла могут быть разными на нескольких серверах. Для файла определен только один вла-

делец – его непосредственный создатель, имеющий права полного доступа к этому файлу. Владелец файла имеет право открыть доступ к своему файлу для других сотрудников предприятия. Разрабатываемый программный продукт использует фиксированную совокупность файлов различных типов, которые загружаются непосредственно с серверов.

Для каждого сотрудника в БД вносится следующая информация:

- табельный номер;
- фамилия, имя, отчество;
- принадлежность к определенному отделу;
- должность;
- рабочий и домашний телефоны.

Отделы характеризуются следующими параметрами:

- название отдела;
- телефон начальника отдела.

Для каждого файла должно быть указано:

- имя владельца;
- название файла;
- дата создания файла.

Серверы имеют следующие параметры:

- название сервера;
- IP-адрес.

Каждый программный продукт имеет свое уникальное имя.

Предполагается, что с БД имеют право работать различные категории пользователей (разграничение этих категорий по правам доступа в данном примере не предусмотрено), которые могут решать следующие основные задачи:

Получение сведений о имеющихся файлах предприятия, в частности:

- полный список всех файлов;
- информация об определенном файле (файлах) в соответствии с критерием запроса.

Получение сведений об определенном файле:

- право доступа сотрудников фирмы к данному файлу;
- расположение файла на сервере (серверах);
- связь с программным продуктом (продуктами);

Получение сведений об отделах фирмы.

Получение сведений о сотрудниках фирмы.

Выдача списка всех имеющихся программных продуктов.

Получение сведений обо всех серверах фирмы.

Кроме того, предусматривается ряд дополнительных возможностей, которые в дальнейшем будем называть запросами к БД:

1. По файлам предприятия*:*

- выдача списка файлов, доступ к которым открыт для нескольких пользователей из разных отделов;

- выдача списка файлов, доступ к которым открыт для сотрудников фиксированного числа отделов;

- выдача списка файлов, пользователями которых являются сотрудники только одного отдела;

- выдача списка файлов, расположенных на разных серверах под одним именем и используемых сотрудниками разных отделов.

2. По программным продуктам предприятия:

- выдача списка программных продуктов, работающих только с одним файлом;

- выдача списка программных продуктов, работающих со всеми серверами;

3. По отделам предприятия:

- выдача списка отделов, сотрудники которых не работают ни с одним файлом;

- выдача списка отделов, сотрудники которых работают со всеми серверами;

4. По серверам предприятия:

- выдача списка серверов, с которыми работают сотрудники фиксированного числа отделов.

5. По сотрудникам предприятия:

- поиск сведений о сотруднике в соответствии с критерием запроса.

Концептуальное моделирование предметной области

Проведенный анализ предметной области позволяет отобразить ее как взаимодействие следующих пяти сущностей: «Файл», «Программа», «Сервер», «Сотрудник», «Отдел».

Сущность «Файл» с точки зрения решаемой задачи должна быть представлена группой свойств (атрибутов), позволяющих однозначно ее характеризовать: имя файла, дата создания и ФИО владельца. Эти атрибуты обусловлены необходимостью выполнения перечисленных выше задач. Однако отдельный экземпляр сущности не будет при этом однозначно идентифицироваться перечисленными атрибутами. Поэтому необходимо добавление еще одного свойства – уникального кода файла, представленного в числовом виде.

Сущность «Программный продукт» характеризуется своим уникальным именем.

Для сущности «Сервер» определим следующие атрибуты: название сервера и его IP-адрес, однозначно идентифицирующие ее.

Сущность «Сотрудник» для обеспечения реализации задач БД должна определяться следующими атрибутами: табельный номер, ФИО, должность, название отдела, телефоны. Табельный номер однозначно определяет каждый экземпляр данной сущности

Сущность «Отдел» определяется названием и телефоном начальника отдела.

Определим теперь имеющиеся между сущностями связи.

Поскольку каждый сотрудник работает в определенном отделе, а в каждом отделе имеется по крайней мере несколько сотрудников, между сущностями «Отдел» и «Сотрудник» имеется связь “один-ко-многим”, которую назовем “работает в”. Данная связь является обязательной со стороны обеих сущностей

Сущности «Сотрудник» и «Файл» связаны отношением “многие-ко-многим” вследствие того, что один и тот же сотрудник может быть владельцем или пользователем совокупности файлов, а каждый файл с различными правами доступа может использоваться рядом сотрудников. Назовем эту связь “использует”. Со стороны сущности “Файл” связь обязательна, а со стороны сотрудник – нет (сотрудники, например, рекламного отдела не принимают участие в разработке программных продуктов)

Аналогичным образом устанавливаем, что между сущностями «Файл» и «Программный продукт» имеется связь “многие-ко-многим”, названная “обеспечивает”; между сущностями «Файл» и «Сервер» также имеется связь “многие-ко-многим” с названием

“хранится”. Эти связи являются обязательными со стороны каждой из сущностей.

Построенная в соответствии с проведенным анализом предметной области *ER*-диаграмма проектируемой БД приведена на рис. п.1.

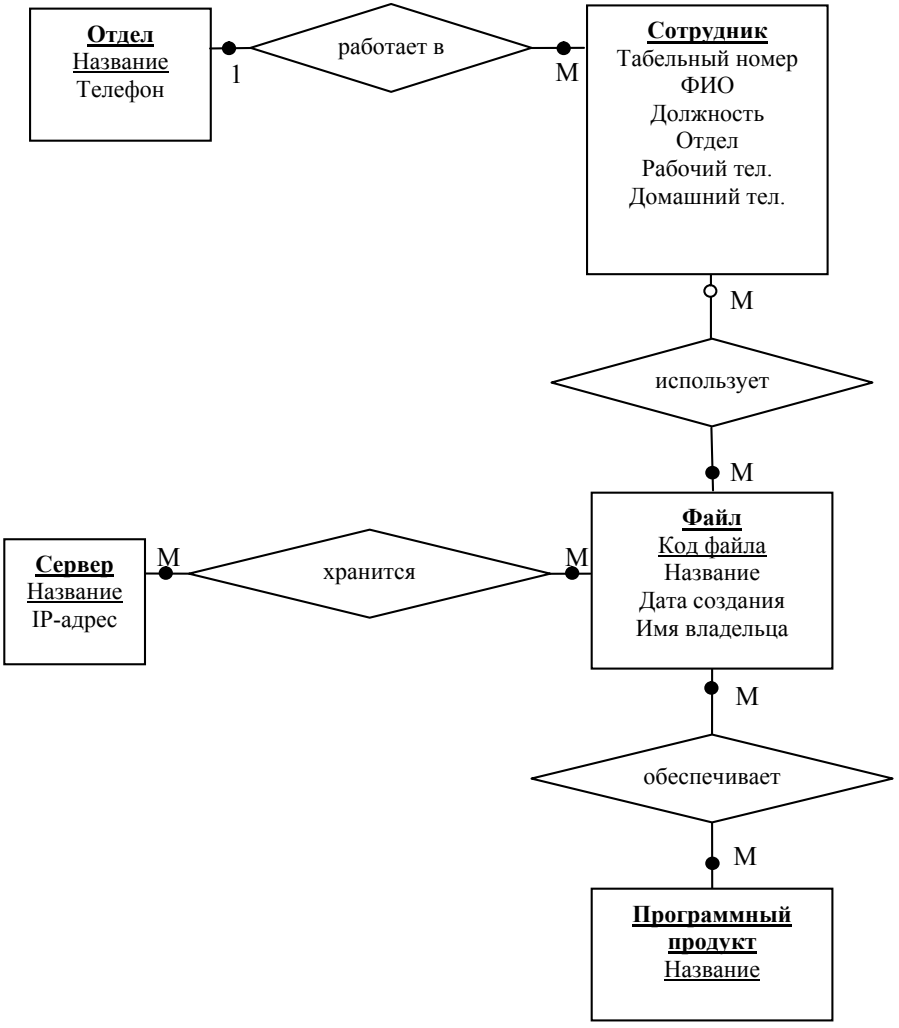


Рис. п.1. ER-диаграмма БД

Построение реляционной модели БД

Преобразование разработанной концептуальной модели БД в реляционную проведем в соответствии со стандартными правилами.

Прежде всего, каждой сущности необходимо поставить в соответствие отношение (таблицу), а каждому атрибуту сущности – атрибут соответствующего отношения (поле таблицы). При этом происходит изменение названий как сущностей, так и самих атрибутов в соответствии с ограничениями используемой в дальнейшем СУБД (так, например, в названиях недопустимо использовать кириллицу, пробелы и некоторые другие символы). Атрибут или атрибуты отношения, однозначно определяющие его кортежи (строки таблицы), становятся первичным ключом соответствующего отношения. Те атрибуты подчиненного отношения, которые определяют связи этого отношения с главным, становятся внешними ключами подчиненного отношения. Для всех атрибутов отношений указываются типы данных в рамках выбранной СУБД, а также обязательность или необязательность задавания определенных значений.

После этого необходимо исключить связи “многие-ко-многим”, поскольку такие связи в реляционной модели недопустимы. С этой целью для каждой пары отношений, между которыми имеется такая связь, необходимо добавить дополнительное отношение, в которое нужно включить два атрибута, являющихся первичными ключами связываемых отношений. В этом же отношении они являются внешними ключами. Такой подход приводит к появлению двух новых связей “многие-ко-многим” между исходными отношениями и дополнительным отношением, что позволяет применять реляционную модель.

Результатом преобразования разработанной *ER*-модели к реляционной является появление приведенных далее отношений.

Вначале приведем список основных отношений, разработанных на стадии проектирования *ER*-модели.

Отношение «File»

Название поля	Тип данных	Ограничения	Ключ
File_ID	Autoincrement	Unique	Primary key
FName	Char[20]	Not null	
FDate	D	Not null	
FVlad	I	Unique	

Отношение «Program»

Название поля	Тип данных	Ограничения	Ключ
Prog_ID	Autoincrement	Unique	Primary key
PName	Char[20]	Not null	

Отношение «Server»

Название поля	Тип данных	Ограничения	Ключ
Serv_ID	Autoincrement	Unique	Primary key
SName	Char[20]	Not null	
SAddress	I	Not null	

Отношение «Sotrudnik»

Название поля	Тип данных	Ограничения	Ключ
Sotr_ID	Autoincrement	Unique	Primary key
SName	Char[40]	Not null	
SDolzhnost	Char[20]	Not null	
SOtdel	I	Unique	Foreign key
SW_phone	Char[15]	Null	
SH_phone	Char[15]	Null	

Отношение «Otdel»

Название поля	Тип данных	Ограничения	Ключ
Otdel_ID	Autoincrement	Unique	Primary key
OName	Char[20]	Not null	
OPhone	Char[15]	Not null	

Далее приводится список вспомогательных отношений, необходимых для реализации связей “многие-ко-многим”.

Отношение «Dostup» создано для реализации связи между отношениями «File» и «Sotrudnik».

Отношение «Dostup»

Название поля	Тип данных	Ограничения	Ключ
Dostup_ID	Autoincrement	Unique	Primary key
DSotr	I	Unique	Foreign key
DFile	I	Unique	Foreign key

Отношение «Keeper» создано для реализации связи между отношениями «File» и «Server».

Отношение «Keeper»

Название поля	Тип данных	Ограничения	Ключ
Keeper_ID	Autoincrement	Unique	Primary key
KFile	I	Unique	Foreign key
KServ	I	Unique	Foreign key

Отношение «Work» создано для реализации связи между отношениями «File» и «Program».

Отношение «Work»

Название поля	Тип данных	Ограничения	Ключ
Work_Id	Autoincrement	Unique	Primary key
WProg	I	Unique	Foreign key
WFile	I	Unique	Foreign key

Создание и редактирование таблиц данных

Решение задачи разработки реляционной модели БД позволяет перейти непосредственно к ее проектированию. Прежде всего нужно создать необходимое количество таблиц – наборов данных, определить их поля и установить свойства полей в соответствии с определенными реляционными отношениями, после чего уже можно устанавливать связи между ними.

Вначале создадим каталог, в котором будет храниться проектируемая БД и управляющее ей приложение, например, **C:\MyDB**. Далее откроем утилиту **Database Desktop** и, выполнив команду **File\Working Directory,** укажим путь к созданному каталогу.

Создание таблиц данных в среде **Delphi** осуществляется с помощью программы (утилиты) **Database Desktop (DBD),** входящей в комплект поставки **Delphi**. Для ее запуска необходимо в главном меню выбрать опцию **Tools\Database Desktop** и далее в меню открывшегося окна – команду **File\New\Table**. При выборе формата таблицы можно согласиться с предлагаемым форматом **Paradox**. Результатом выполнения этих операций явится появление окна макета новой таблицы, в котором необходимо определить параметры ее полей. На рис. п.2 в качестве примера приводится структура полей таблицы **File**.

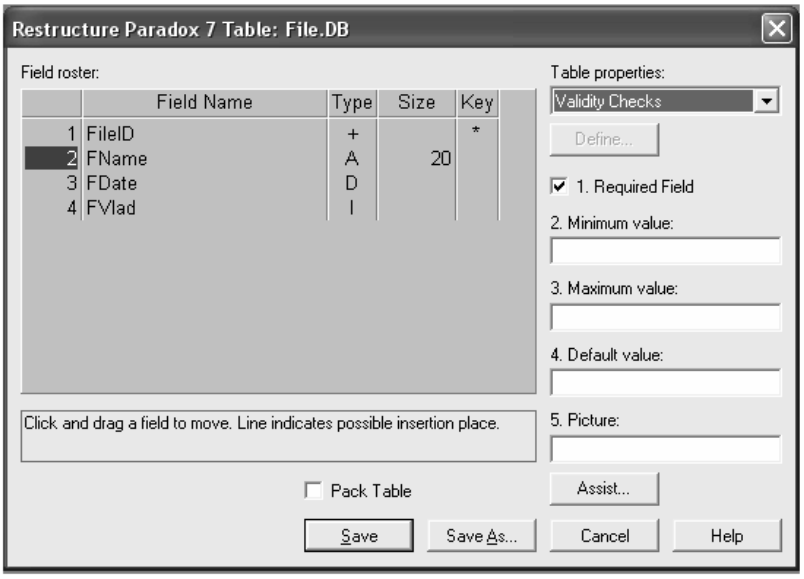


Рис. п.2. Структура полей таблицы **File**

Обратите внимание, что поля **FName** и **FDate** помечены как **Required Field**, т. е. обязательны для заполнения.

Следующим этапом работы является заполнение полей созданных таблиц.

*Примечание***.** На самом деле при проектировании реальной БД этот этап может быть пропущен – можно сразу установить связи между таблицами и приступить к разработке управляющего БД приложения (СУБД), под управлением которого внесение информации в БД будет выполнено впоследствии при автоматическим контроле корректности всех связей. Однако в рамках конструирования учебной БД выполнение данного этапа полезно для лучшего уяснения принципов отношений в соответствии с постановкой задачи (следующим из анализа предметной области).

Способов заполнения таблиц данными может быть много, однако имеется ряд правил, выполнение которых является обязательным. Ниже приводится один из возможных вариантов.

Рассмотрим таблицы **Otdel, Sotrudnik, File** и **Dostup** (рис. п.3).

Предположим, что на предприятии имеется 5 отделов – операционный, центральный, отдел тестирования, отделы звука и видео.

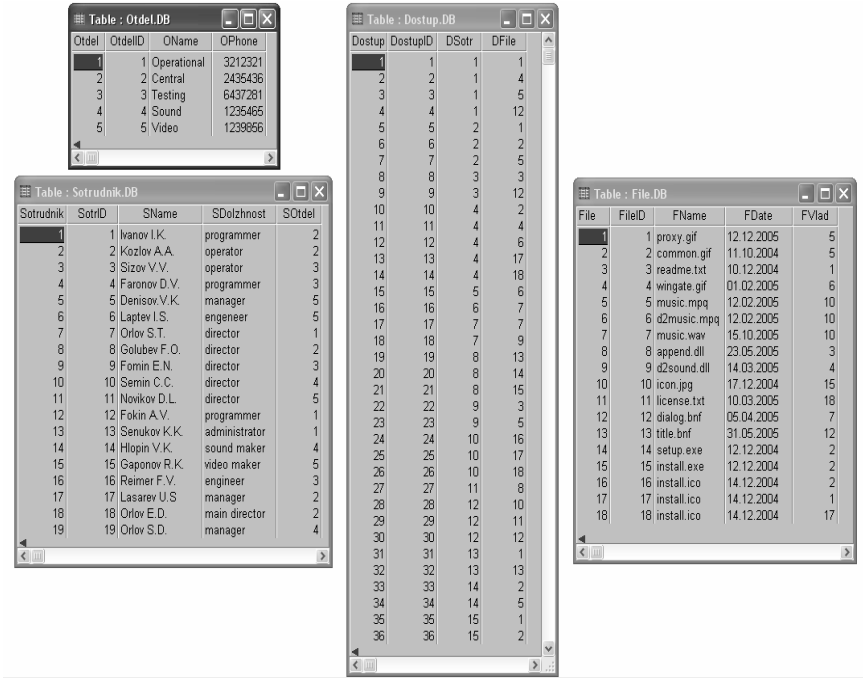


Table : Otdel.DB

Otdel	OtdelID	OName	OPhone
1	1	Operational	3212321
2	2	Central	2435436
3	3	Testing	6437281
4	4	Sound	1235465
5	5	Video	1239856

Table : Sotrudnik.DB

Sotrudnik	SotrID	SName	SDolzhnost	SOtdel
1	1	Ivanov I.K.	programmer	2
2	2	Kozlov A.A.	operator	2
3	3	Sizov V.V.	operator	3
4	4	Faronov D.V.	programmer	3
5	5	Denisov.V.K.	manager	5
6	6	Laptev I.S.	engeneer	5
7	7	Orlov S.T.	director	1
8	8	Golubev F.O.	director	2
9	9	Fomin E.N.	director	3
10	10	Semin C.C.	director	4
11	11	Novikov D.L.	director	5
12	12	Fokin A.V.	programmer	1
13	13	Senukov K.K.	administrator	1
14	14	Hlopin V.K.	sound maker	4
15	15	Gaponov R.K.	video maker	5
16	16	Reimer F.V.	engineer	3
17	17	Lasarev U.S	manager	2
18	18	Orlov E.D.	main director	2
19	19	Orlov S.D.	manager	4

Table : Dostup.DB

Dostup	DostupID	DSotr	DFile
1	1	1	1
2	2	1	4
3	3	1	5
4	4	1	12
5	5	2	1
6	6	2	2
7	7	2	5
8	8	3	3
9	9	3	12
10	10	4	2
11	11	4	4
12	12	4	6
13	13	4	17
14	14	4	18
15	15	5	6
16	16	6	7
17	17	7	7
18	18	7	9
19	19	8	13
20	20	8	14
21	21	8	15
22	22	9	3
23	23	9	5
24	24	10	16
25	25	10	17
26	26	10	18
27	27	11	8
28	28	12	10
29	29	12	11
30	30	12	12
31	31	13	1
32	32	13	13
33	33	14	2
34	34	14	5
35	35	15	1
36	36	15	2

Table : File.DB

File	FileID	FName	FDate	FVlad
1	1	proxy.gif	12.12.2005	5
2	2	common.gif	11.10.2004	5
3	3	readme.txt	10.12.2004	1
4	4	wingate.gif	01.02.2005	6
5	5	music.mpq	12.02.2005	10
6	6	d2music.mpq	12.02.2005	10
7	7	music.wav	15.10.2005	10
8	8	append.dll	23.05.2005	3
9	9	d2sound.dll	14.03.2005	4
10	10	icon.jpg	17.12.2004	15
11	11	license.txt	10.03.2005	18
12	12	dialog.bnf	05.04.2005	7
13	13	title.bnf	31.05.2005	12
14	14	setup.exe	12.12.2004	2
15	15	install.exe	12.12.2004	2
16	16	install.ico	14.12.2004	2
17	17	install.ico	14.12.2004	1
18	18	install.ico	14.12.2004	17

Рис. п.3. Таблицы **Otdel, Sotrudnik, File** и **Dostup**

Для того, чтобы каждого сотрудника предприятия закрепить за определенным отделом, в таблице **Sotrudnik** имеется поле **SOtdel**, являющееся внешним ключом этой таблицы, связывающие ее записи с записями в таблице **Otdel** отношением “многие-к-одному”. В качестве значений поля **SOtdel** выступают возможные значения поля **OtdelID** таблицы **Otdel.** Очевидно, что сама схема построения данных таблиц исключает возможность возникновения ситуации, когда один сотрудник может быть закреплен более чем за одним отделом. Прочитать информацию из связанных таким образом таблиц легко. Например, сотрудниками отдела тестирования являются Сизов В.В., Фаронов Д.В. и Реймер Ф.В.

Таблица **Dostup** служит для связи между таблицами **Sotrudnik** и **File**, разбивая связь “многие-ко-многим” между этими таблицами

на две связи “один-ко-многим”, и заполняется следующим образом. Поле **DSotr** этой таблицы служит внешним ключом для ее связи с таблицей **Sotrudnik**. Количество записей с одинаковым значением данного поля соответствует количеству файлов, доступ которым имеет сотрудник с соответствующим значением в поле **SotrID** в таблице **Sotrudnik**. Значения в поле **DFile** (внешний ключ для связи с таблицей **File**) определяют файлы, с которыми работает данный сотрудник. Так, например, Козлов А.А. имеет доступ к файлам **proxy.gif, common.gif**, **readme.txt** и **music.mpq**. В обратном направлении с файлом dialog.bnf могут работать сотрудники Иванов И.К., Сизов В.В. и Фокин А.В. В поле **FVlad** указывается непосредственный владелец данного файла, разрешающий доступ к нему остальным сотрудникам.

Оставшиеся четыре таблицы (рис. п.4) заполняются аналогичным образом.

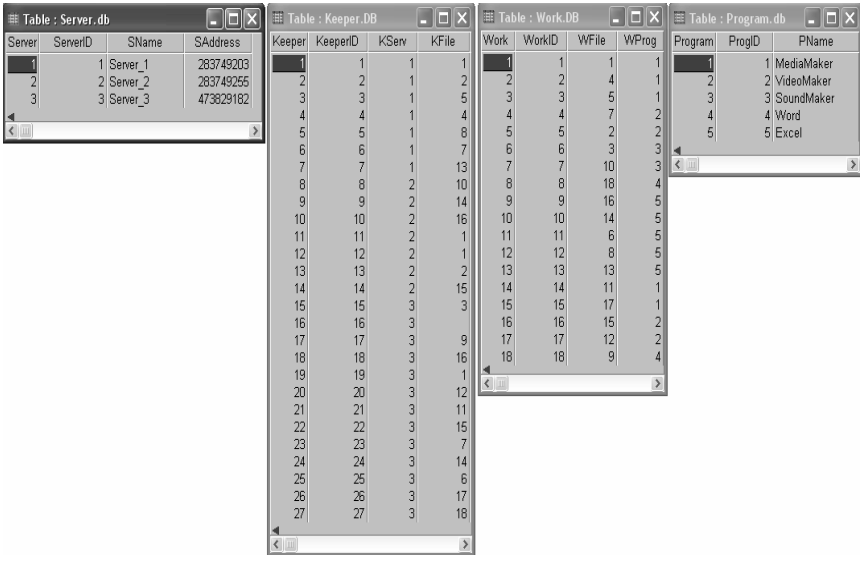


Table : Server.db

Server	ServerID	SName	SAddress
1	1	Server_1	283749203
2	2	Server_2	283749255
3	3	Server_3	473829182

Table : Keeper.DB

Keeper	KeeperID	KServ	KFile
1	1	1	1
2	2	1	2
3	3	1	5
4	4	1	4
5	5	1	8
6	6	1	7
7	7	1	13
8	8	2	10
9	9	2	14
10	10	2	16
11	11	2	1
12	12	2	1
13	13	2	2
14	14	2	15
15	15	3	3
16	16	3	
17	17	3	9
18	18	3	16
19	19	3	1
20	20	3	12
21	21	3	11
22	22	3	15
23	23	3	7
24	24	3	14
25	25	3	6
26	26	3	17
27	27	3	18

Table : Work.DB

Work	WorkID	WFile	WProg
1	1	1	1
2	2	4	1
3	3	5	1
4	4	7	2
5	5	2	2
6	6	3	3
7	7	10	3
8	8	18	4
9	9	16	5
10	10	14	5
11	11	6	5
12	12	8	5
13	13	13	5
14	14	11	1
15	15	17	1
16	16	15	2
17	17	12	2
18	18	9	4

Table : Program.db

Program	ProgID	PName
1	1	MediaMaker
2	2	VideoMaker
3	3	SoundMaker
4	4	Word
5	5	Excel

Рис. п.4. Таблицы **Server, Keeper, Work** и **Program**

Структура таблиц **Keeper** и **Work** аналогична структуре таблицы **Dostup**, все файлы размещаются на трех серверах и поддерживаются с помощью пяти приложений.

Создание связей между таблицами

Теперь можно приступить к созданию связей между таблицами в соответствии с разработанной схемой . Рассмотрим эту процедуру на примере таблиц **File** и **Sotrudnik**. Связь между этими таблицами осуществляется с помощью вспомогательной таблица **Dostup**, в которой поля **DFile** и **DSotr** являются внешними ключами для таблиц **File** и **Sotrudnik** соответственно и должны выступать в качестве вторичных ключей при образовании связей.

Необходимо открыть утилиту **Database Desktop**, далее открыть таблицу **Dostup** и выполнить команду **Restructure**. В списке свойств таблицы **Table Properties** нужно выбрать пункт **Referential Integrity** и далее выбрать опцию **Define**. Свяжем вначале таблицы **Dostup** и **File**. В окне **Fields** выбираем поле **DFile** и перемещаем его в окно **Chield Fields**, в правом – таблицу **file.db** и нажимаем на кнопку со стрелкой влево. Сохраним созданную связь под именем **DostupFile** (рис. п.5).

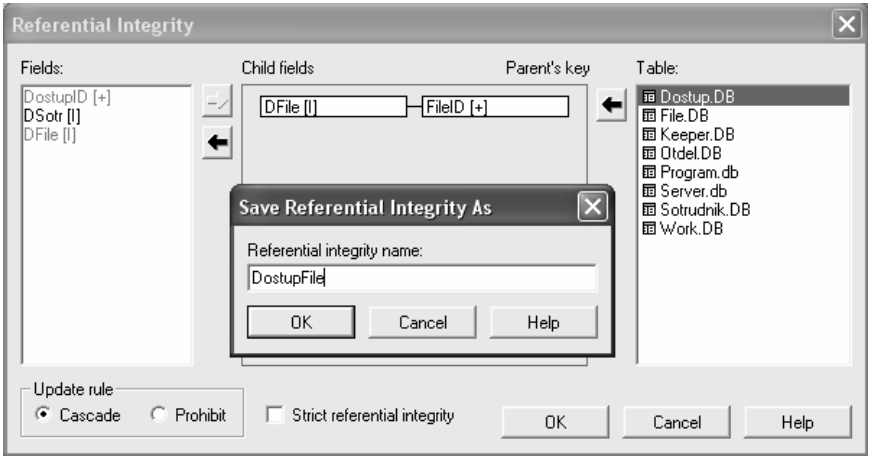


Рис. п.5. Окно создания связи между таблицами **File** и **Sotrudnik**

Аналогичным образом свяжем таблицы **Dostup** и **Sotrudnik** по полям **DSotr** и **SotrID** под именем **DostupSotrudnik**.

Необходимо сохранить сделанные изменения. Если теперь вновь выполнить команду **Restructure** и списке **Table Properties**

выбрать пункт **Secondary Indexes**, то можно увидеть, что поля **DFile** и **DSotr** стали вторичными индексами (рис. п.6).

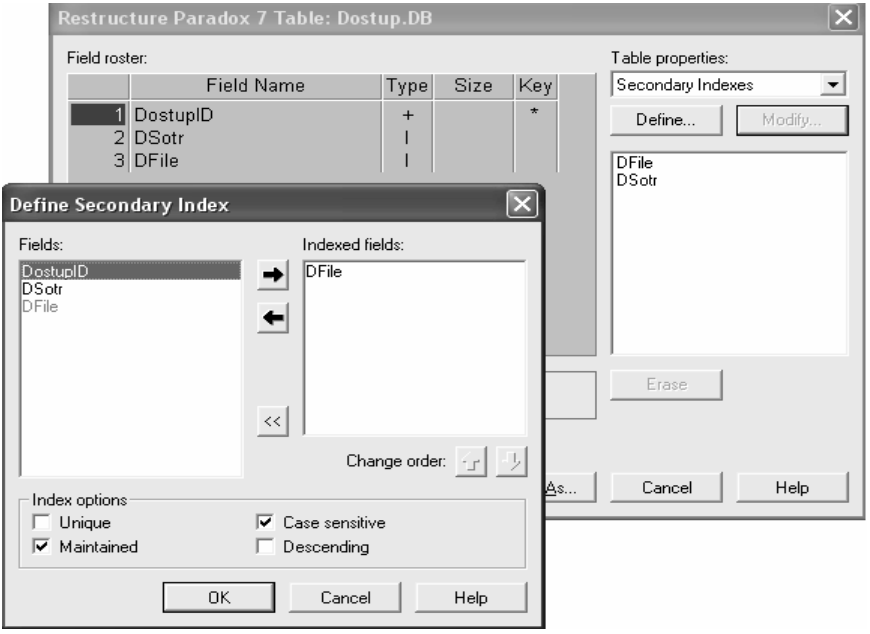


Рис. п.6. Окна, отображающие связи между таблицами по вторичным индексам

Аналогичные действия нужно проделать с таблицами **Keeper** и **Work**, сохранив изменения под следующими именами.

Chield Table	Parent Table	Referential Integrity Name	Secondary Index Name
Keeper	File Server	KeeperFile KeeperServer	KFile KServ
Work	File Program	WorkFile WorkProgram	WFile WProg

Можно убедиться в том, что вторичные индексы создались в соответствии с проделанным.

Разработка приложения

Прежде всего следует создать в среде Delphi новый проект и сохранить его в рабочий каталог разрабатываемой БД. После этого с помощью инструмента **SQL Explorer** необходимо создать псевдоним БД (ее название или имя) и указать путь к каталогу БД.

На начальной стадии проектирования приложения, обеспечивающего работу с БД, необходимо наличие двух окон – головного окна программы (компонент **Form**) и модуля данных (**DataModule**), на котором будут размещены компоненты **Table** и **DataSource**. Для создания модуля данных в главном меню проекта нужно выбрать опцию **File\New\Others** и в открывшемся окне объектов в разделе **New** выбрать компонент **Data Module** В свойстве **Name** этого компонента запишем **DM,** сохраним его в рабочий каталог БД под именем **Data** и свяжем его с приложением, используя команду **Project\Add to project.**

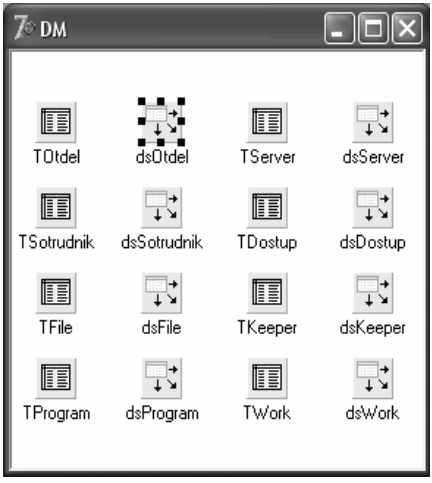


Рис. п.7. Окно модуля **DM**

Разместим в окне модуля данных 8 компонентов **Table** и столько же компонентов **DataSource** (рис. п.7). Для компонентов **Table** необходимо определить значения свойств **DataBaseName, TableName** и **Name,** сославшись на имя БД, имя таблицы БД и создав имя нового объекта приложения. Значение свойства **Active** зададим равным **true**. Для компонентов **DataSource** изменим значения свойств **Name** и свяжем каждый из них с соответствующим компонентов **Table**, изменяя свойство **DataSet**.

Примечание. Значения свойств **Name** компонентов **Table** и **DataSource** можно не изменять, однако для облегчения дальнейшей работы удобно в их названиях использовать названия таблиц БД.

Теперь нужно создать связи между таблицами **File, Server, Program** и **Sotrudnik** с помощью таблиц **Dostup, Keeper** и **Work**. Для этого следует перейти на вкладку **Diagram** модуля данных **Data**, после чего переместить туда все таблицы из окна дерева объектов модуля **DM**.

Для начала создадим связь между перечисленными таблицами по вторичным индексам **DFile, KFile** и **WFile** (рис. п.8).

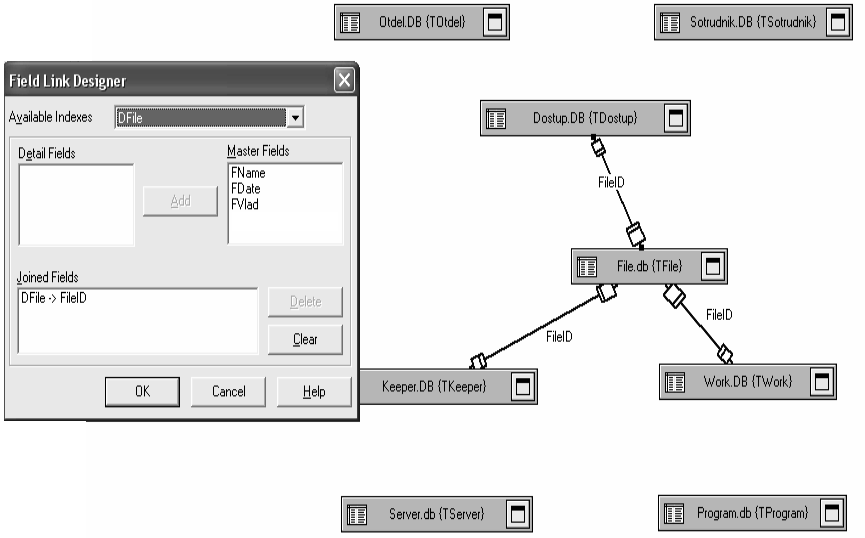


Рис. п.8. Окно модуля с тремя связями по вторичным ключам

После этого, несмотря на то, что таблицы **Otdel, Sotrudnik, Server** и **Program** пока остаются несвязанными, вернемся к конструированию главного окна приложения. Обеспечение просмотра наборов данных может быть реализовано различными способами с использованием как одного, так и нескольких компонентов **DBGrid**. При этом возможные виды связей между таблицами также могут различаться. Выбор того или иного способа зависит, в частности, от дополнительных условий, определяемых функциональными задачами практического использования проектируемого приложения.

В нашем случае предлагается следующий вариант. Из имеющихся 8 наборов данных 5 представляют собой основные сущности

в соответствии с разработанной *ER*-моделью, а 3 являются вспомогательными, служащими для возможности реализации отношений “многие-ко-многим” между таблицей **File** (главная) и таблицами **Server, Program** и **Sotrudnik** (подчиненные).

Для просмотра основных и дополнительных таблиц будем использовать по одному компоненту **DBGrid**. Просмотр дополнительных таблиц, казалось бы, вовсе необязателен. Однако, поскольку они связаны с таблицей **File**, одновременная визуализация этих таблиц и таблицы **File** позволяет получить дополнительную информацию, не используя таких средств, как например, запросы. В самом деле, перемещаясь между строками таблицы **File**, можно сразу же получить ответ на вопросы, на каком сервере хранится данный файл, какими программами он поддерживается и какие пользователи имеют к нему доступ.

Для перехода между наборами данных, отображаемых с помощью компонентов **DBGrid**, будем использовать компонент **MainMenu**.

Разместим на главном окне приложения объекты **DBGrid1, DBGrid2** и **MainMenu** и создадим пункты главного меню так, как это показано на рис. п.9 (пункт “Опции” понадобится в дальнейшем). Кроме этого, разместим на главном окне два объекта **Label**, которые будут отображать названия таблиц.

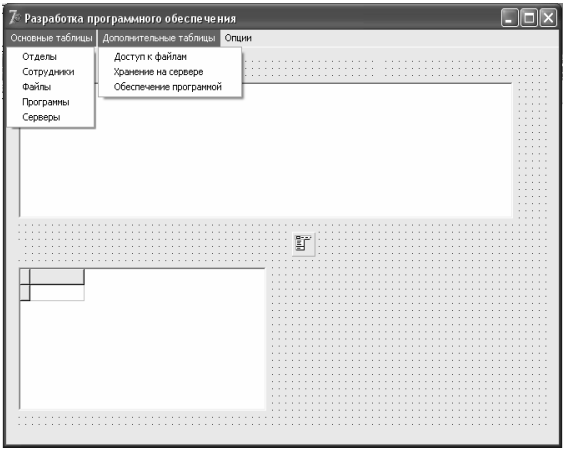


Рис. п.9. Вид главного окна приложения

Ниже приводятся процедуры обработки событий при выборе в меню пунктов “**Отделы**” и “**Доступ к файлам**”.

```
procedure TForm1.N2Click(Sender: TObject); // выбор в меню пункта “Отдел”
begin
DBGrid1.Visible:=true;
DBGrid1.DataSource:=DM.dsOtdel; // привязка сетки к нужному набору данных
DBGrid2.Visible:=false; // дополнительные наборы данных отображать не нужно
N7.Enabled:=false; // пункт меню “Дополнительные таблицы” недоступен
Label1.Visible:=true;
Label1.Caption:='Список отделов'; // заголовок набора данных
Label2.Visible:=false;
end;

procedure TForm1.N8Click(Sender: TObject); // выбор пункта “Доступ к файлам”
begin
DBGrid2.Visible:=true;
DBGrid2.DataSource:=DM.DSDostup;
Label2.Visible:=true;
Label2.Caption:='Доступ сотрудников к файлам';
end;
```

Аналогично программируются остальные пункты меню.

На этом начальную стадию разработки приложения можно считать завершенной. Остается откомпилировать проект, сохранить все изменения и запустить его на исполнение.

Сразу же становится ясно, что работать с наборами данных достаточно сложно: заголовки столбцов читать неудобно, нужная информация во многих случаях недоступна. Например, при выборе набора данных из таблиц **File** и **Dostup** (рис. п.10) вместо фамилии сотрудника-владельца данного файла в поле **FVlad** отображается целое число – значение первичного ключа таблицы **Sotrudnik**, в

поле **DSotr** вместо фамилий сотрудников, имеющих доступ к данному файлу, также отображается значение первичного ключа таблицы **Sotrudnik**. Кроме того, поля **DostupID** и **DFile** не несут никакой информации, поэтому их отображение не является необходимым.

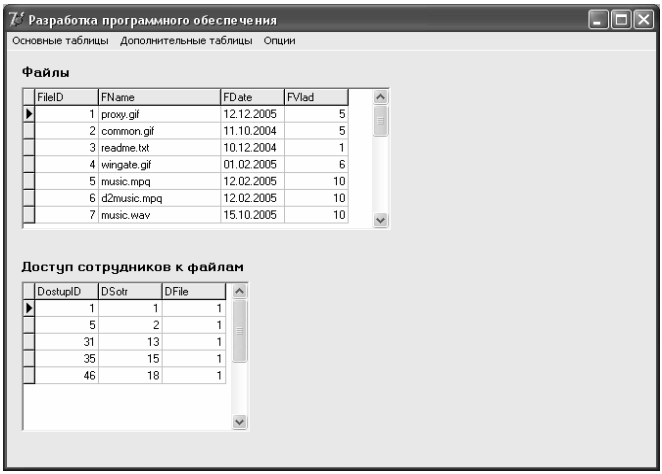


Рис. п.10. Вид главного окна после запуска приложения на исполнение

Все эти неудобства можно преодолеть с помощью создания так называемых подстановочных полей. Для этого нужно активизировать окно модуля данных, выберать в нем таблицу **TDostup** и двойным щелчком левой кнопки мыши вызвать окно создания объектов-полей. Сначала выполняется команда **Add all fields**, а затем – команда **New Field**. В появившемся окне следует установить такие свойства нового поля, как это показано на рис. п.11.

Теперь в данном поле будут отображаться фамилии сотрудников из таблицы **Sotrudnik**, имеющие доступ к файлу, для которого одно и то же значение поля **DFile** таблицы **Dostup** соответствует различным значениям поля **DSotr** этой же таблицы.

Все остальные поля таблицы **Dostup**, являющиеся неинформативными, можно скрыть, выбирая в окне их свойств **Object Inspector** значение **false** для свойства **Visible**.

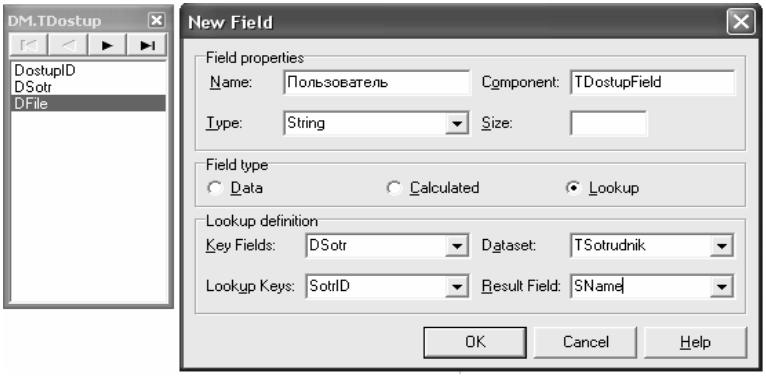


Рис. п.11. Окна создания объектов-полей

Аналогичным образом создается подстановочное поле в таблице **File** для того, чтобы вместо номера сотрудника отображалась его фамилия. Поле **FVlad** можно скрыть, а заголовки полей **FileID**, **FName** и **FDate** изменить, изменяя значения свойств **DisplayLabel**.

После проделанных изменений внешний вид главного окна приложения после запуска его на исполнение должен измениться так, как это показано на рис. п.12.

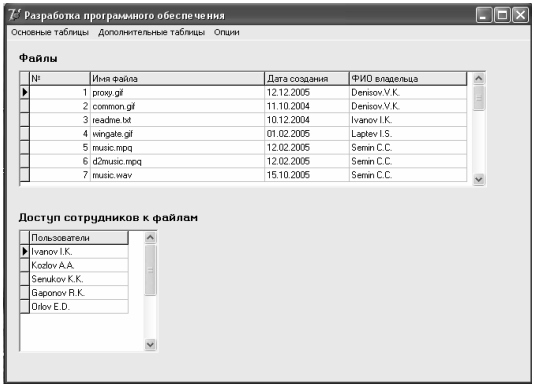


Рис. п.12. Вид главного окна приложения

Если теперь перейти в окно модуля данных **Diagram**, то можно увидеть (рис. п.13), что, во-первых, появились новые выделенные курсивом подстановочные поля, во-вторых, по этим полям возникли новые связи между таблицами, отличающиеся своим внешним видом и цветом от ранее созданных связей по вторичным индексам.

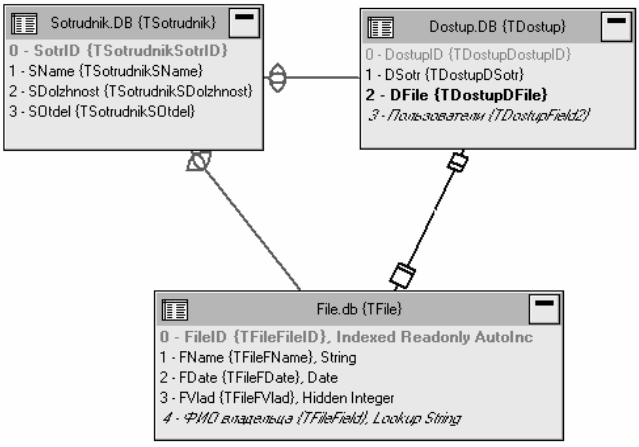


Рис. п.13. Окно модуля данных **Diagram**

После создания объектов-полей для остальных таблиц, добавления необходимого количества подстановочных полей, исходя из очевидных соображений удобства работы с наборами данных, окно модуля данных **Diagram** примет вид, изображенный на рис. п.14.

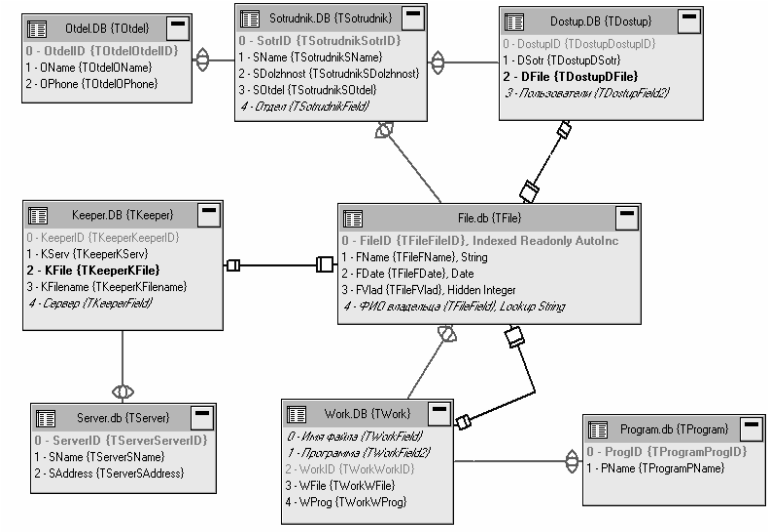


Рис. п.14. Окно модуля данных **Diagram** после установления всех необходимых связей

Запросы к БД

Для того, чтобы излишне не загромождать главное окно приложения компонентами, создадим для запросов отдельное окно **Zapros** и разместим в этом окне по одному компоненту **Button** и **DBGrid**, 2 компонента **Label**, 9 компонентов **RadioButton**, а также компоненты **Query** и **DataSourse**. Свойства **Caption** компонентов **Button1, Zapros, RadioButton** и **Label1** нужно изменить так, как это показано на рис. п.15.

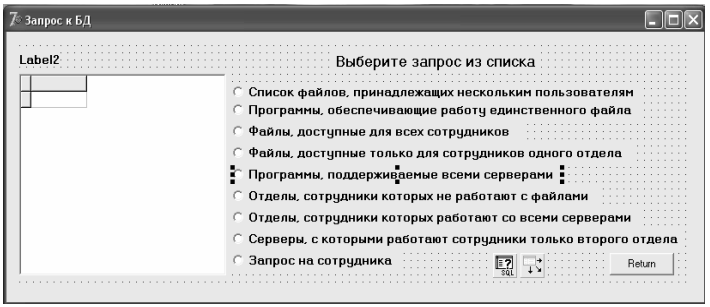


Рис. п.15. Окно запросов к БД

Для компонента **DataSourse** в качестве значения свойства **DataSet** укажем **Query1**, а для компонента **DBGrid** в качестве значения свойства **DataSource** укажем **DataSource1**.

Кроме того, в свойстве **AutoSize** окна **Zapros** установим значение **True**.

Для вызова данного окна можно создать в главном меню проекта обработчик события **Опции\Запрос**, запрограммировав его следующим образом:

```
procedure TForm1.N12Click(Sender: TObject);
begin
Zapros.Visible:=true;
Zapros.DBGrid1.Visible:=false;
Zapros.Label1.Visible:=true;
Zapros.Label2.Visible:=false;
end;
```

Реализация запросов осуществляется при возникновении события **OnClick** для компонентов **RadioButton**.

Рассмотрим два примера.

А) Список файлов, доступ к которым имеют все сотрудники

```
procedure TZapros.RadioButton3Click(Sender: TObject);
begin
Label1.Visible:=false;
Label2.Visible:=true;
Label2.Caption:='Файлы, доступные для всех сотрудников';
Radiobutton1.Visible:=false;
Radiobutton2.Visible:=false;
Radiobutton3.Visible:=false;
Radiobutton4.Visible:=false;
Radiobutton5.Visible:=false;
Radiobutton6.Visible:=false;
Radiobutton7.Visible:=false;
Radiobutton8.Visible:=false;
Radiobutton9.Visible:=false;
DBGrid1.Visible:=true;
Radiobutton3.Checked:=false;
with Query1 do begin
      Close;
      SQL.Clear;
            SQL.Add('SELECT File_ID as код, FName as файл');
      SQL.Add('FROM "file", "dostup", "sotrudnik", "otdel"');
      SQL.Add('WHERE');
      SQL.Add(' DFile=File_ID and DSotr=Sotr_ID  ');
      SQL.Add('GROUP BY File_ID, FName ');
      SQL.Add('HAVING ((COUNT(distinct SOtdel)=3))');
      Open;
      if RecordCount <> 0
      then DataSource1.DataSet := Query1
      else ShowMessage('В БД нет записей, удовлетворяющих' +
       #13 + 'критерию запроса'); // #13 – переход к новой строке
   end;
end;
```

Результатом выполнения данного запроса является возникновение приблизительно следующего окна (рис. п.16):

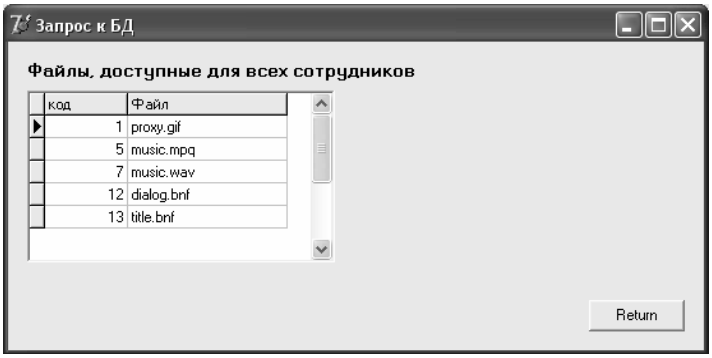


Рис. п.16. Вид окна после реализации запроса

Б) Запрос на сотрудника

В качестве критерия используется **ФИО** сотрудника. Результатом выполнения запроса является информация о том, в каком отделе этот сотрудник работает.

```
procedure TZapros.RadioButton9Click(Sender: TObject);
var buf:string[30];
begin
buf := InputBox('Выборка информации по  сотрудникам',
                'Введите Фамилию сотрудника ','');
Label1.Visible:=false;
Label2.Visible:=true;
Label2.Caption:='Запрос на сотрудника';
Radiobutton1.Visible:=false;
Radiobutton2.Visible:=false;
Radiobutton3.Visible:=false;
Radiobutton4.Visible:=false;
Radiobutton5.Visible:=false;
Radiobutton6.Visible:=false;
Radiobutton7.Visible:=false;
Radiobutton8.Visible:=false;
Radiobutton9.Visible:=false;
```

```
DBGrid1.Visible:=true;
Radiobutton9.Checked:=false;
   if buf = '' then exit;
   with Query1 do begin
      Close;
      SQL.Clear;
      SQL.Add('SELECT OName as Отдел, SName as ФИО');
      SQL.Add('FROM "sotrudnik", "otdel" ');
      SQL.Add('WHERE');
            SQL.Add('Sotdel=Otdel_ID and Sotr_ID in (select
         Sotr_ID from "sotrudnik" where (SName like '"' + buf +
         '%' + '"'))');
      Open;
      if RecordCount <> 0
      then
      begin
      DataSource1.DataSet := Query1;
      DBGrid1.Columns.Items[0].Width:=150;
      DBGrid1.Columns.Items[1].Width:=130;
      end
      else ShowMessage('По запросу ничего не найдено');
   end;
end;
```

Результат выполнения данного запроса приведен на рис. п.17.

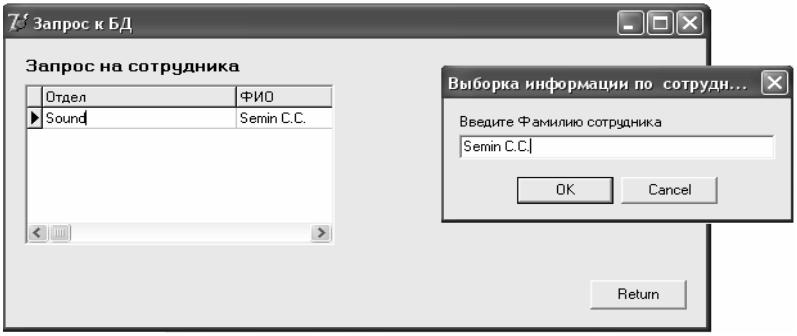


Рис. п.17. Результат выполнения запроса

Поиск записей в БД

В данном разделе с использованием метода **Filter** будет реализован механизм, позволяющий осуществлять поиск записей в БД в соответствии с определенными критериями.

Задача № 1. Нужно разработать метод, который позволяет отыскивать в БД файлы либо по названию, либо по дате создания, либо одновременно по этим двум свойствам. Для этого воспользуемся методом **Filter**. Прежде всего нужно создать новое окно, в котором будут задаваться критерии фильтрации. Для этого добавим в проект новую форму, свяжем ее с проектом, в окне ее свойств **Object Inspector** установим значения **FilterF** для свойства **Name** и **Фильтр** для свойства **Caption**. Далее разместим в этом окне по 2 компонента **Label** и **Edit** и один компонент **BitBtn** из раздела **Additional** палитры компонентов. Для свойства **Kind** этого компонента следует установить значение **bkOK**. После этого вновь созданное окно примет приблизительно следующий вид (рис. п.18):

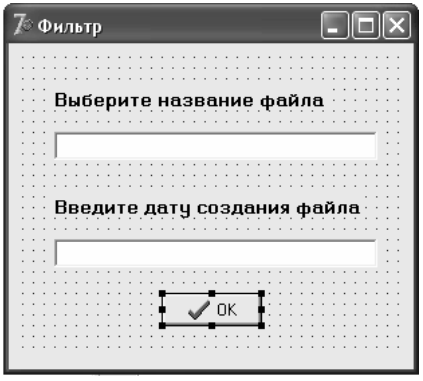


Рис. п.18. Вид окна для реализации поиска в БД

Для вызова этого окна в меню главного окна проекта добавим новую кнопку **Опции\Фильтр**.

Алгоритм реализации поставленной задачи поиска файлов следующий. Процедура фильтрации возникает при открытом наборе данных таблицы **File** в случае нажатия на кнопку главного меню **Фильтр** и начинается с открытия соответствующего окна:

```
procedure TForm1.N13Click(Sender: TObject);
begin
if DBGrid1.DataSource=DM.DSFile then
     FilterF:=TFilterF.Create(Self);
end;
```

В этом окне вводятся данные об искомом файле либо в оба поля **Edit1** и **Edit2**, либо только в какое-нибудь одно из них. После этого при нажатии на кнопку **OK** происходит сравнение введенных данных с данными из таблицы **File**. Например, если осуществляется поиск по названию файла, операция сравнения выглядит следующим образом:

```
Filter:=concat('FName=', FilterF.Edit1.Text);
```

Далее должно возникнуть событие **OnFilterRecord**, для чего нужно установить значение **true** для свойства **Filtered** таблицы **File**.

```
procedure TForm1.N13Click(Sender: TObject);
begin
if DBGrid1.DataSource=DM.DSFile then
begin
FilterF:=TFilterF.Create(Self);
if FilterF.ShowModal=mrOK then
if (FilterF.Edit1.Text<>'') and (FilterF.Edit2.Text='') then
begin
FilterF.Edit1.Text:=chr(39)+FilterF.Edit1.Text+chr(39);
with DM,TFile do
begin
Filter:=concat('FName=',FilterF.Edit1.Text);
Filtered:=true;
end;
end;
if (FilterF.Edit2.Text<>'') and (FilterF.Edit1.Text='') then
begin
FilterF.Edit2.Text:=chr(39)+FilterF.Edit2.Text+chr(39);
with DM,TFile do
begin
Filter:=concat('FDate=',FilterF.Edit2.Text);
Filtered:=true;
end;
end;
if (FilterF.Edit1.Text<>'') and (FilterF.Edit2.Text<>'') then
begin
FilterF.Edit1.Text:=chr(39)+FilterF.Edit1.Text+chr(39);
```

```
    FilterF.Edit2.Text:=chr(39)+FilterF.Edit2.Text+chr(39);
    with DM,TFile do
    begin
    Filter:=concat('(FName=',FilterF.Edit1.Text,')and
(','FDate=',FilterF.Edit2.Text,')');
    Filtered:=true;
    end;
    end;
    if (FilterF.Edit1.Text='') and (FilterF.Edit2.Text='') then
DM.TFile.Filtered:=false;
    end;
    end;
```

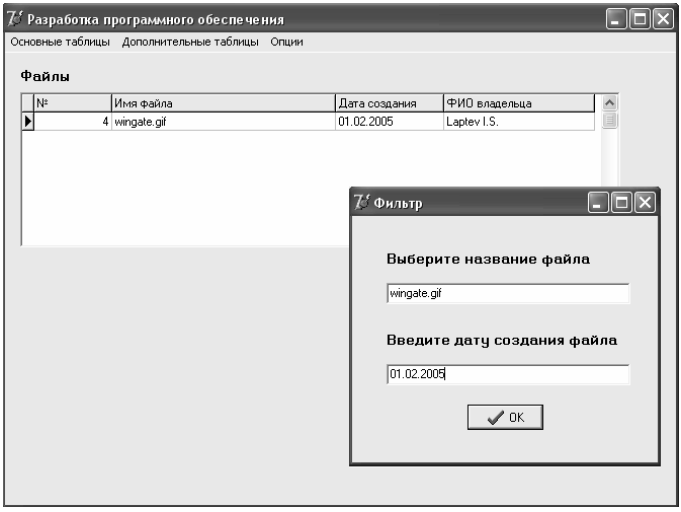


Рис. п.19. Реализация поиска нужного файла

В тексте данной процедуры учтено, что поскольку значения полей **Edit** являются символьными, имя файла и дату вводить в них нужно в одинарных кавычках. Автоматическое добавление кавычек с использованием символа одинарной кавычки **chr**(39) облегчает ввод данных и позволяет избегать ошибок (рис. п.19).

Другими методами поиска данных являются методы **Lookup** и **Locate**

Создание отчетов

Так же как и запросы к БД, отчет служит инструментом для представления части данных, хранимых в БД, в соответствии с определенным образом заданными критериями. В данном разделе для создания отчетов рассмотрим технологию **Rave Reports** (вкладка **Rave**), разработанную для версии Delphi 7. Эту технологию используем для создания трех типов отчетов – отчет, составленный по данным одной таблицы, отчет типа главный-детальный, составленный по данным двух таблиц – главной и подчиненной, а также сгруппированный отчет, составленный по данным из нескольких таблиц, объединенных каким-либо одним общим свойством.

Отчет по данным одной таблицы (на примере таблицы File)

Для создания отчета прежде всего необходимо поместить на главное окно проекта компонент **RVTableConnection**, находящийся на вкладке **Rave**, и связать его с набором данных **File**, изменив свойство **Table**. Все дальнейшие действия осуществляются с помощью утилиты **Rave Reports Designer**, вызываемой из главного меню **Delphi** командой **Tools/Rave Designer**.

Сначала нужно связать новый отчет с источником данных, который он будет представлять. Для этого необходимо выполнить команду **File/New Data Object**, далее в открывшемся окне выберать опцию **Direct Data View**, затем нажать на кнопку **Next** и выделить компонент **RVTableConnection1**. Для завершения создания связи с набором данных **File** нужно выполнить команду **Finish**.

Дальнейшая разработка отчета осуществляется с использованием мастера создания отчетов, вызываемого командой **Tools/ReportWizards/SimpleTable**. На первом шаге следует выбрать источник данных **DataView1**, на втором – определить список отображаемых полей. Для этого нужно установить флажки для полей **Имя файла, Дата создания, ФИО владельца**. На третьем шаге можно изменить порядок следования полей. На четвертом предлагается изменить заголовок отчета и установить ширину полей – это можно сделать сейчас, а можно впоследствии, когда будет сформирован макет страницы отчета. Наконец, на заключительном шаге можно изменить шрифты заголовка и отображаемых полей. Если на

последних шагах изменения внесены не были, то после выполнения команды **Generate** следует ожидать появление следующего макета отчета (рис. п.20):

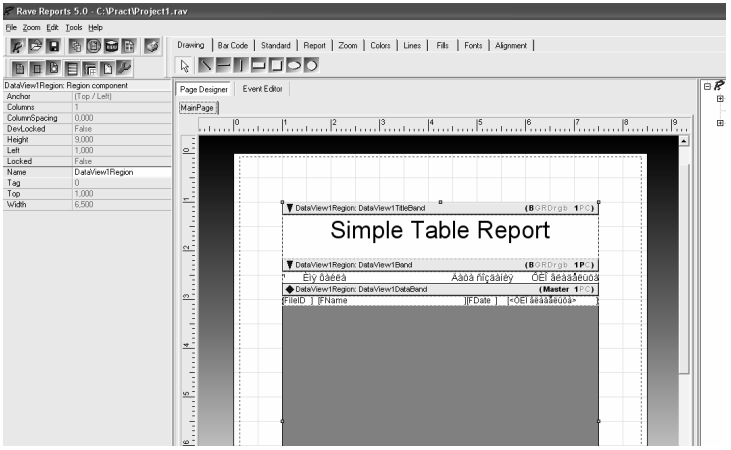


Рис. п.20. Окно макета отчета

Изменяя свойства названий и шрифтов заголовков, положения их на странице, можно привести окно отчета примерно к такому виду (рис. п.21):

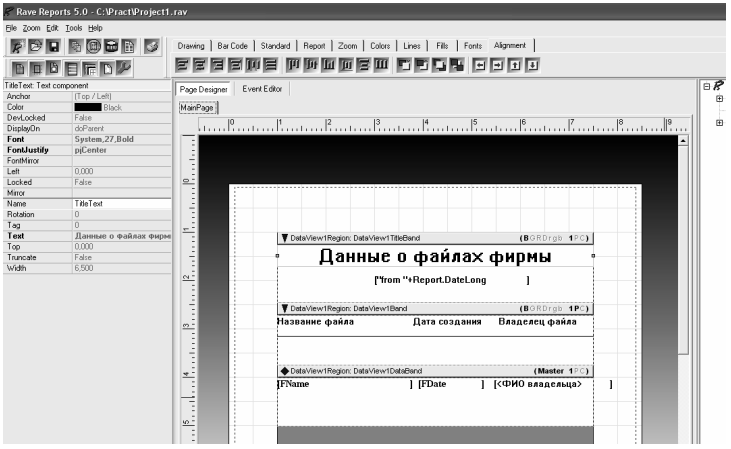


Рис. п.21. Окно отчета

Обратите внимание, что для автоматической генерации даты отчета под заголовком вставлена функция **Report.DateLong**.

Теперь необходимо связать созданный отчет с приложением. Для этого следует поместить на главное окно приложения компонент **RvProject1** с вкладки **Rave**, в свойстве **ProjectFile** указать файл отчета **Project1.rav**, в главном меню добавить обработчик события **Опции/Отчеты/по файлам** и запрограммировать его, например, следующим образом:

```
procedure TForm1.N15Click(Sender: TObject);
begin
if DBGrid1.DataSource=DM.DSFile then
RvProject1.Execute;
end;
```

Отчет "главный-детальный"

В случае реализации такого типа отчета в нем оказываются представленными данные из записи главной таблицы и все данные из подчиненной таблицы, связанные с этой записью. Создадим такой отчет для таблиц **Otdel** (главная) и **Sotrudnik** (подчиненная).

Так же, как и для предыдущего типа отчета, утилита **Rave Designer** содержит мастер создания отчета "главный-детальный", который вызывается командой **Tools/Master/DetailReport**. Однако гораздо удобнее создать отчет вручную, используя следующую последовательность шагов:

Шаг 1. На главное окно приложения необходимо поместить еще два компонента **TvTableConnection**, в свойстве **Table** указать на таблицы **Otdel** и **Sotrudnik**, после чего запустить утилиту **RaveDesigner**.

Шаг 2. В проекте нового отчета необходимо создать два объекта **Direct DataView1** и **DataView2**, используя команду **File/NewDataObject**. Эти компоненты нужно связать с соответствующими компонентами **TvTableConnection2** и **TvTableConnection3**.

Шаг 3. С вкладки **Report** нужно поместить на макет страницы отчета **Page Designer** компонент **Region**, назначение которого заключается в том, что именно на нем будут в дальнейшем располагаться все остальные отображающие данные из таблиц компонен-

ты. Поэтому компонент **Report** следует максимально “растянуть” на всю область страницы отчета.

Шаг 4. На вкладке **Report** имеются компоненты **Band** и **DataBand**, служащие для отображения заголовков и самих данных из таблиц соответственно. Эти компоненты следует поместить на компонент **Region** один под другим в следующем порядке:

1. Band – для заголовка отчета. На нем нужно расположить компонент **Text** с вкладки **Standard** для задания заголовка отчета.

2. DataBand – для отображения данных из таблицы **Otdel**. В свойстве **DataView** необходимо сослаться на объект **DataView1**. На нем разместить компоненты **Text** для подзаголовка и **DataText** (вкладка **Report**) для отображения самих данных из таблицы **Otdel**. В свойстве **Text** компонента **Text** следует установить значение **“Название отдела”.** Компонент **DataText** свяжем с объектом **DataView1**, изменяя соответствующее свойство. Поскольку по условию задачи компонент **DataText** предназначен для отображения списка имеющихся отделов, то в его свойстве **DataField** необходимо указать название соответствующего поля **OName**.

Шаг 5. Теперь перейдем к формированию данных из таблицы **Sotrudnik**. Для этого на компонент **Region** ниже поместим еще один компонент **Band**, а на него – компонент **Text**. В качестве свойства **BandStyle** компонента **Band** выберем **Body Header**, а в свойстве **Text** компонента **Text** запишем **“Сотрудники”.** Далее разместим ниже еще один компонент **DataBand** и настроим его таким образом, чтобы с его помощью правильно отображались данные из подчиненной таблицы **Sotrudnik**.

В свойстве **DataView** необходимо сослаться на объект **DataView2**, а в свойстве **DetailKey** - на поле **SOtdel**, являющимся внешним ключом таблицы **Sotrudnik**, которое как раз и обеспечивает связь между таблицами **Sotrudnik** и **Otdel**. В свойстве **MasterDataView** нужно сделать ссылку на главную таблицу, выбрав объект **DataView1**, а в свойстве **MasterKey** сослаться на первичный ключ **OtdelID** таблицы **Otdel**. Проведенные действия позволили установить требуемую связь между таблицами для правильного отображения набора данных.

После этого в свойстве **ControllerBand** компонента **DataBand2** необходимо установить значение **DataBand1**, сославшись таким

образом на соответствующий компонент, отображающий данные из главной таблицы, а для свойства **BandStyle** выбрать значение **Detail**. Далее, изменяя свойство **ControllerBand** компонента **Band2**, нужно сослаться на компонент **DataBand2**.

В заключение разместим на компоненте **DataBand2** компонент **DataText**, в его свойстве **DataView** укажем **DataView2**, а в свойстве **DataField** – **SName**.

Макет страницы отчета должен иметь вид, изображенный на рис. п.22.

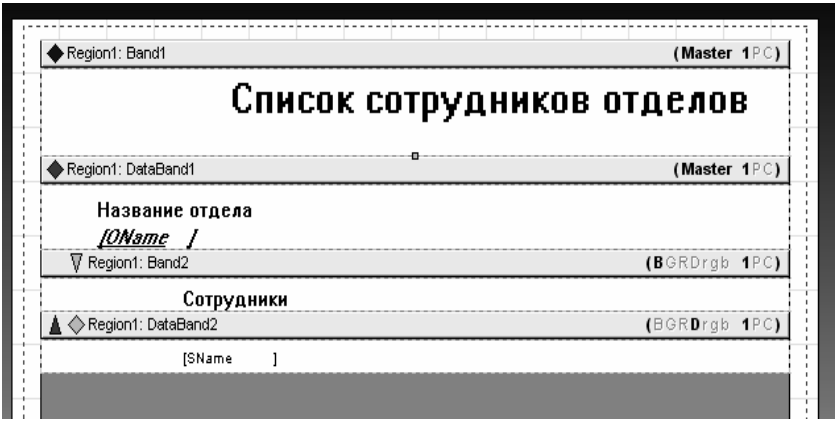


Рис. п.22. Макет страницы отчета

Реализация отчета осуществляется размещением на главном окне проекта компонента **RvProject2**, в свойстве которого **ProjectFile** устанавливается значение **Project2.rav**, добавлением в главное меню обработчика события **Опции/Отчеты/по сотрудникам** и программированием его, например, следующим образом:

```
procedure TForm1.N16Click(Sender: TObject);
begin
if DBGrid1.DataSource=DM.DSSotrudnik then
RvProject2.Execute;
end;
```

Окно отчета имеет вид, изображенный на рис. п.23.

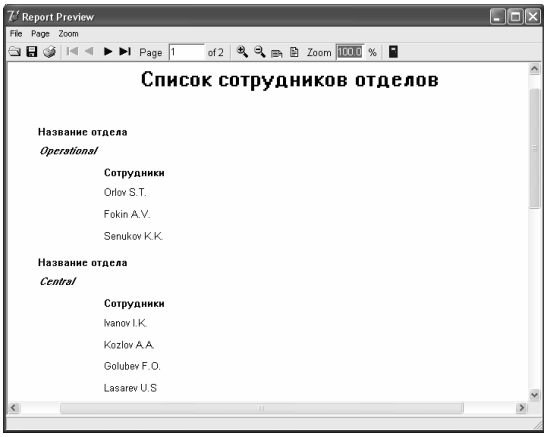


Рис. п.23. Реализация отчета

Сгруппированный отчет

Данный тип отчета используется тогда, когда некую информацию, содержащуюся в БД, необходимо разделить на группы данных, объединенных каким-либо одним общим свойством. Реализация такого отчета осуществляется с помощью изменения свойств **GroupView** и **GroupKey** компонентов **DataBand**.

Составим отчет, в котором все сотрудники сгруппированы по занимаемым ими должностям.

Разместим на главном окне проекта компоненты **Query1** и **RvQueryConnection1**. В свойстве последнего **Query** установим значение **Query1**. В свойстве **DataBaseName** компонента **Query1** укажем псевдоним БД, а в свойстве **SQL** запишем следующий текст запроса:

SELECT SName, SDolzhnost
FROM Sotrudnik
GROUP BY SDolzhnost , SName

После этого в свойстве **Active** данного компонента установим значение **true**.

Теперь нужно открыть утилиту **Rave Designer** и создать новый объект **Direct Data View**, связав его с компонентом **RvQueryConnection1**.

Далее необходимо поместить на макет страницы отчета компонент **Region** и разместить на нем последовательно компоненты **Band1** (заголовок отчета), **Band2** (заголовок групп отчета, объединяющих сотрудников по должностям) и **DataBand1** (непосредственно для вывода фамилий сотрудников), после чего можно приступать к настройке их свойств.

Шаг 1. На компоненте **Band1** разместим компонент **Text** и в его свойстве **Text** укажем название отчета.

Шаг 2. В свойствах компонентов **Band2** и **DataBand1 GroupDataView** и **GroupKey** укажем **DataView1** и **SDlozhnost** соответственно.

Шаг 3. На компоненте **Band2** разместим 2 компонента **Text** для отображения заголовков групп отчета и один компонент **DataText**, в свойстве **DataView** которого нужно сослаться на объект отображения данных **DataView1**, а в свойстве **DataField** укажем значение поля **SDolzhnost**. Для ссылки на компонент **DataBand1** следует изменить соответствующим образом свойство **ControllerBand** компонента **Band2**.

Шаг 4. На компонент **DataBand1** поместим компонент **DataText** и настроим его свойства таким образом, чтобы с помощью этого компонента отображались фамилии сотрудников.

После этого макет страницы отчета должен приобрести приблизительно следующий вид (рис. п.24):

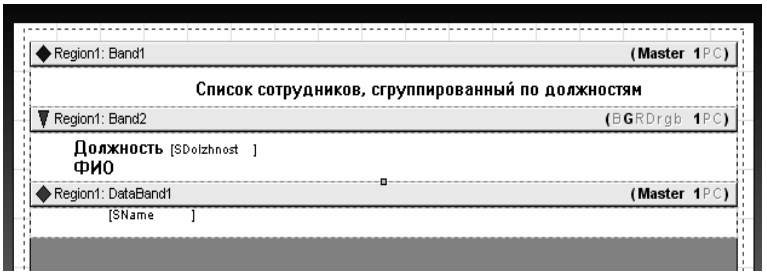


Рис. п.24. Макет страницы отче-

Подключение отчета к приложению осуществляется с помощью еще одного компонента **RvProject** и обработчика события в главном меню проекта. Если все сделано правильно, при вызове отчета он должен иметь следующий вид (рис. п.25):

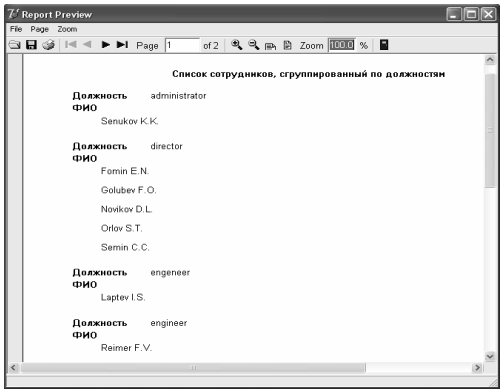


Рис. п.25. Реализация отче-

Модификация БД

В этой части разработки проекта рассмотрим возможности внесения изменений в БД – удаления и добавления записей в таблицы, изменения их содержания. При разработке соответствующих методов необходимо помнить о том, что каждое внесенное в БД изменение не должно нарушать ее целостность и непротиворечивость. Для этого следует учитывать прежде всего имеющиеся связи между таблицами. Так, удаление записи из таблицы **File**, являющейся главной по отношению к таблицам **Dostup, Server** и **Program**, повлечет за собой нарушение соответствующих связей, поскольку ряд записей в последних трех таблицах останутся “бесхозными”, что мгновенно приведет к сбою в работе приложения, восстановить который окажется сложной задачей.

Рассмотрим вначале методы, позволяющие вносить изменения в пять основных таблиц БД – **File, Otdel, Sotrudnik, Server** и **Program**.

Для осуществления операций модификации БД используем контекстное меню **PopupMenu**, вызываемое нажатием на правую кнопку мыши. Поместим на главное окно проекта данный компонент с вкладки **Standard** и создадим в нем три обработчика событий – изменить содержание текущей записи, добавить запись, удалить запись (рис. п.26):

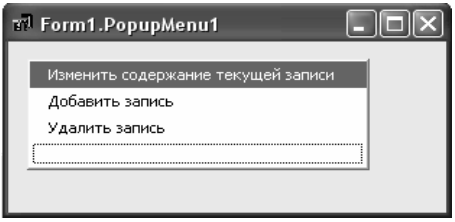


Рис. п.26. Окно контекстного меню

Для того, чтобы это меню было доступно только для интересующих нас таблиц, нужно в свойстве компонента **DBGrid1 PopupMenu**, как раз и предназначенного для их отображения, сослаться на объект **PopupMenu1**. Теперь приступим к реализации процедур модификации.

Изменение содержания записей

Для того, чтобы иметь возможность изменять содержание записей, хранящихся в таблице **File**, удобно добавить к проекту еще одну форму (с названием **Files**), на которую поместить 3 компонента **Label**, по два компонента **BitBtn** и **DBEdit**, а также один компонент **DBLookupCombobox** (рис. п.27).

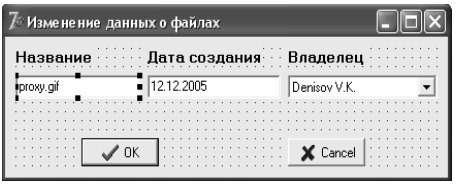


Рис. п.27. Окно для изменения записей БД

Компоненты **DBEdit** и **DBLookupCombobox** нужно связать с набором данных **File**. Для этого необходимо установить в их свойствах **DataSource** и **DataField** соответствующие ссылки, предварительно включив новую форму в проект.

Далее запрограммируем кнопку **“Изменить”** контекстного меню следующим образом. Для внесения изменений в БД будем использовать метод кэширования, заключающийся в том, что с целью уменьшения вероятности возникновения ошибки при проведении операции модификации БД приложение создает локальную копию данных (кэш) для того, чтобы все сделанные изменения записывались вначале в нее, а не в реальную БД.

```
procedure TForm1.N19Click(Sender: TObject);
begin
if DBGrid1.DataSource=DM.DsFile then
begin
```

```
Files:=TFiles.Create(Self);
with DM,TFile do
begin
CachedUpdates:=true;
Edit;
if (Files.ShowModal=mrOK) then
begin
Post;
ApplyUpdates;
end
else
begin
Cancel;
CancelUpdates
end;
CachedUpdates:=false;
Files.Free
end;
end;
end;
```

Теперь при открытой таблице **File** нажатие на кнопку "**Изменить**" в ее текущей строке появится окно новой формы, содержащее данные из этой строки. Название файла и дату его создания можно при этом менять произвольно, а имя владельца для избежания ошибки предлагается выбрать из списка сотрудников.

Аналогичным образом разрабатываются процедуры для изменения данных, содержащихся в оставшихся четырех таблицах.

Добавление новой записи

Для обеспечения возможности добавления новых записей в таблицу **File** можно воспользоваться созданным окном **Files**, соответствующим образом запрограммировав обработчик события "**Добавить запись**" контекстного меню:

```
procedure TForm1.N20Click(Sender: TObject);
begin
if DBGrid1.DataSource=DM.DsFile then
```

```
begin
with DM,TFile do
begin
Append;
Files:=TFiles.Create(Self);
if Files.ShowModal=mrOK then Post else Cancel;
Files.Free
end;
end;
end;
```

Необходимо помнить, что при добавлении таким образом новой записи в таблицу, являющуюся родительской по отношению к таблицам **Dostup, Keeper** и **Work**, эта запись пока еще никак не связана с записями в подчиненных таблицах, что легко можно увидеть, запустив приложение на исполнение. Таким образом, новая запись частично является "бесхозной" – новый файл имеет владельца, однако он не относится ни к какому серверу и не поддерживается ни одной программой. Кроме того, ни один из сотрудников фирмы, кроме непосредственного владельца, не имеет возможности доступа к данному файлу.

Удаление записей

Так же, как для операций изменения и добавления записей, рассмотрим процесс удаления записи на примере таблицы **File**. При реализации этой операции также необходимо помнить о связях между таблицами. Одним из основных принципов построения реляционных БД является поддержка ссылочной целостности между отношениями, обеспечивающая поддержание непротиворечивого состояния БД в любой момент времени. Это означает, что при удалении записей из главной таблицы все записи подчиненных таблиц, связанные с ней, должны удаляться автоматически. В противном случае в подчиненных таблицах окажутся ни с чем не связанные записи. Однако в среде Delphi этого автоматически не происходит, вследствие чего выполнение команды **Delete** приведет к возникновению исключительной ситуации, ведущей к сбою программы (рис. п.28):

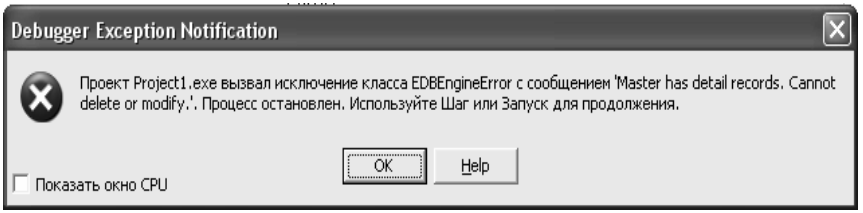


Рис. п.28. Окно предупреждения о возникновении исключительной ситуации

Предотвратить такую ситуацию следует процедурным способом. Будем исходить из того, что прежде чем удалить запись из главной таблицы, следует вначале удалить все записи из подчиненных таблиц, связанные с ней. В противном случае в выполнении операции удаления будет отказано.

Одним из возможных вариантов реализации данного алгоритма может быть следующий обработчик события "**Удалить запись**" контекстного меню:

```
procedure TForm1.N21Click(Sender: TObject);
var
LookupResult: Variant;
child:integer;
CanDelete: boolean;
str:string;
begin
child:=0;
if DBGrid1.DataSource=DM.DsFile then
with DM,TFile do
begin
child:=0;
LookupRe-
sult:=DM.TDostup.Lookup('Dfile',TFileFileID.Value,'Dfile');
if not (VarType(LookupResult)in [varNull]) then child:=child+1;
LookupRe-
sult:=DM.TKeeper.Lookup('Kfile',TFileFileID.Value,'Kfile');
if not (VarType(LookupResult)in [varNull]) then child:=child+1;
LookupRe-
sult:=DM.TWork.Lookup('Wfile',TFileFileID.Value,'Wfile');
```

```
if not (VarType(LookupResult)in [varNull]) then child:=child+1;
if child=0 then Delete
else ShowMessage('Данный файл не может быть удален!'+#13+'Файл связан с пользователем, программой или сервером');
end;
end;
```

Здесь введена вспомогательная переменная **child**. Функция **Lookup** последовательно осуществляет поиск в подчиненных таблицах записей, связанных с удаляемой. При нахождении таких записей переменная **child** меняет значение, что приводит к отказу в удалении с появлением соответствующего сообщения.

Другие операции модификации БД

В этом разделе изложены некоторые дополнительные способы внесения изменений в разработанную БД.

БД спроектирована таким образом, что у каждого файла имеется единственный владелец, обладающий полным доступом к этому файлу и обеспечивающий возможность доступа другим сотрудникам. Предположим, что в процессе деятельности возникает необходимость в доступе к какому-либо файлу новому сотруднику либо в отказе доступа для определенного сотрудника. Реализуем поставленную задачу следующим образом.

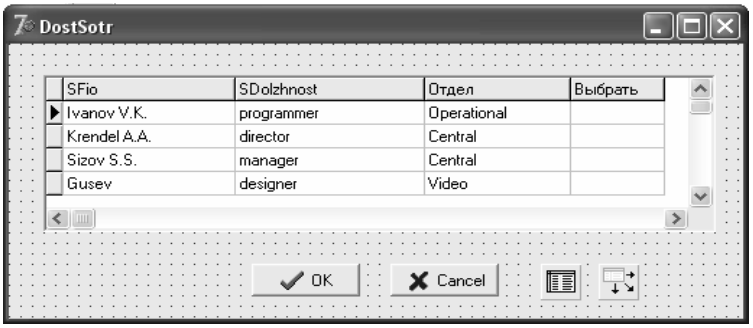


Рис. п.29. Окно новой формы

Создадим в проекте новую форму и свяжем ее с проектом. Разместим на ней один компонент **DBGrid**, два компонента **BitBtn** и по одному компоненту **Table** и **DataSource**.

Назовем новую форму **DostSotr**. В свойстве **Kind** двух кнопок установим значения **OK** и **Cancel**. В свойстве **Name** компонента **Table** установим значение **TPersonal** и свяжем его с таблицей **Sotrudnik**. Компонент **DBGrid** нужно настроить таким образом, чтобы в нем отображался список сотрудников из соответствующей таблицы БД (рис. п.29).

Для отображения списка отделов (столбец "**Отдел**") в таблице **TPersonal** создано подстановочное поле. Столбец "**Выбрать**" добавлен к списку столбцов компонента **DBGrid** вручную.

Для вызова нового окна в головном окне проекта создается обработчик события "**Разрешить доступ к файлу**" в разделе "**Опции**" главного меню:

```
if DBGrid2.DataSource=DM.dsDostup then
begin
with DM, TDostup  do
begin
DostSotr:=TDostSotr.Create(Self);
end;
end;
```

В этой процедуре при добавлении нового сотрудника начинается работа с таблицей **Dostup**, поскольку именно эта таблица связывает таблицы **File** и **Sotrudnik**.

Теперь начнем программирование окна **DostSotr**.

Для начала запрограммируем обработчик события **OnCreate**, возникающий при открытии этого окна:

```
procedure TDostSotr.FormCreate(Sender: TObject);
begin
TPersonal.Open;
List:=TStringList.Create;
List.Sorted:=true;
end;
```

Переменную **List** объявим в разделе **Public** для того, чтобы она была доступна другим модулям проекта. Именно в этой переменной будет храниться информация о том сотруднике, которого нужно добавить к списку:

```
public
```

List: TStringList;
{ Public declarations }

Пусть программа будет разработана так, чтобы для добавления нового сотрудника к имеющемуся списку необходимо нажать левую кнопку мыши в поле "**Выбрать**" в соответствующей строке и нажать на кнопку **OK** (рис. п.30).

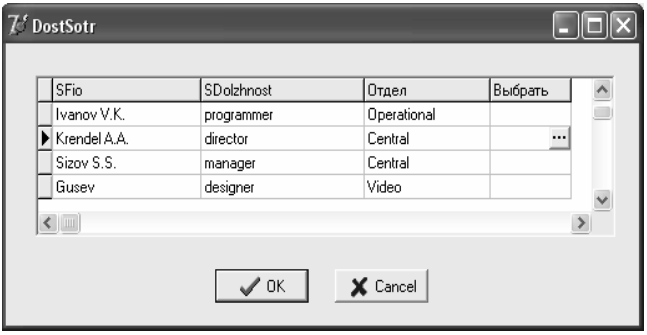


Рис. п.30. Добавление нового сотрудника

Для программирования щелчка мыши в поле "**Выбрать**" понадобятся две новые процедуры: **OnDrowColumnCell** и **OnEditButtonClick** компонента **DBGrid**. Эти процедуры выполняются в момент нажатия на кнопку **cbsEllipsis**, появляющуюся в ячейке при щелчке на ней левой кнопкой мыши в случае, если в свойстве **ButtonStyle** столбца **DBGrid1.Columns[номер столбца]** установлено значение **cbsEllipsis**. Первая процедура позволяет прорисовывать изображения в ячейке компонента **DBGrid**, используя для этого стандартные методы **Canvas**, и служит для визуального отображения выбранных сотрудников в текущей строке, а помощью процедуры **OnEditButtonClick** будем отбирать нужных сотрудников.

Обработчики событий **OnDrowColumnCell** и **OnEditButtonClick** могут быть запрограммированы так:

```
procedure TDostSotr.DBGrid1DrawColumnCell(Sender: TObject;
const Rect: TRect; DataCol: Integer; Column: TColumn;
State: TGridDrawState);
begin
if (DataCol=7) and List.Find(inttostr(TPersonalSotr_ID.Value),k)
then
```

```
with DBGrid1.Canvas do
begin
s:='+';
if not(gdSelected in State) then font.Color:=clRed;
TextOut(Rect.Right-Textwidth(s)-2,Rect.Top+2,s)
end;
end;

procedure TDostSotr.DBGrid1EditButtonClick(Sender: TObject);
var  k:integer;
begin
TPersonal.IndexFieldNames:='Sotr_ID';
if not List.Find(inttostr(TPersonalSotr_ID.Value),k) then
begin
List.Sorted:=false;
List.Add(TPersonalSotr_ID.AsString);
k:=List.Count-1;
end;
List.Sorted:=true;
end;
```

Целочисленная переменная **k** служит счетчиком выбранных сотрудников. Условие на значение **List.Find(inttostr(TPersonalSotr_ID.Value),k)** позволяет избегать повторного добавления одного и того же сотрудника при повторном нажатии на кнопку в одной и той же ячейке. В переменной **List** накапливаются первичные ключи добавленных сотрудников посредством команды **List.Add(TPersonalSotr_ID.AsString)**.

Для многократного использования созданного метода необходимо предусмотреть обнуление списка **List** после завершения работы с текущим окном. Для этого можно, например, следующим образом запрограммировать закрытие окна:

```
procedure TDostSotr.FormDestroy(Sender: TObject);
begin
List.Free;
end;
```

Кроме того, можно предупредить ситуацию, когда пользователь отобрал нужных сотрудников и нажал на кнопку **Cancel**. Для этого можно использовать метод **OnCloseQuery**:

```
procedure TDostSotr.FormCloseQuery(Sender: TObject; var Can-
Close: Boolean);
begin
if not IsBtnOK and (List.count>0) then
CanClose:=MessageDlg('Вы действительно хотите закрыть окно
'+
'с потерей всех отобранных пользователей?', mtError,
[mbOK,mbCancel],0)=mrOK
else CanClose:=true;
end;
```

Логическую переменную объявим там же, где и переменную **List**. После этого закрытие окна будет сопровождаться следующим предупреждением (рис. п.31):

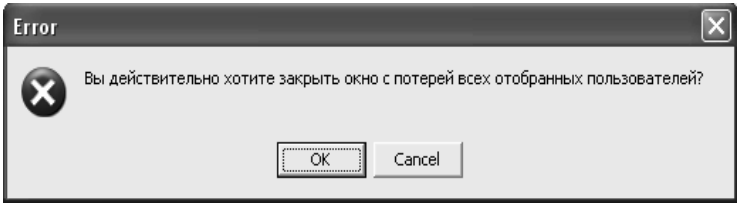


Рис. п.31. Окно предупреждения о возможной потере всех дан-

На этом программирование окна **DostSotr** заканчивается.

Теперь остается завершить программирование обработчика “**Разрешить доступ к файлу**” на главном окне проекта.

```
procedure TMain.N20Click(Sender: TObject);
var
k:integer;
LookupResult: Variant;
begin
if DBGrid2.DataSource=DM.dsDostup then
begin
with DM, TDostup  do
begin
```

```
DostSotr:=TDostSotr.Create(Self);
if DostSotr.ShowModal=mrOK then
begin
// если выбрали сотрудников
if DostSotr.List.count>0 then
begin
// временно отключаем связь с родительской таблицей
// для возможности внесения изменений в дочернюю
TDostup.MasterSource.Enabled:=false;
for k:=0 to DostSotr.List.Count-1 do
begin
LookupResult:=DM.TDostup.Lookup('Dsotr;Dfile',
VarArrayOf([DostSotr.List[k],TFileFile_ID.Value]),'Dsotr;Dfile');
//добавляем новую запись только, если она еще не встречалась
if (VarType(LookupResult)in [varNull]) then
begin
// вносим изменения
Append;
TDostupDfile.Value:=TFileFile_ID.Value;
TDostupDsotr.AsString:=DostSotr.List[k];
Post;
end
else Cancel;
end;
end;
end
else  Cancel;
end;
DostSotr.Free;
end;
// восстанавливаем связи между таблицами
DM.TFile.Active:=true;
DM.TDostup.MasterSource.Enabled:=true;
end;
```

Основы трехзвенной архитектуры

В данном разделе изучаются основы технологии трехзвенной архитектуры проектирования СУБД. Такая архитектура включает в себя три компонента – сервер БД, сервер приложений и клиента. В архитектуре такого типа сервер приложений, являясь промежуточным звеном между клиентом и сервером БД, берет на себя основные функции по реализации управления БД. При этом сервер приложений располагается либо на компьютере сервера БД, либо на отдельном компьютере. Что касается клиента, то его функции сводятся к минимуму и заключаются в передаче серверу приложений запроса к БД. Всю работу по поиску информации в БД сервер приложений берет на себя и затем передает ее обратно клиенту. Поэтому доступ клиента к информации, содержащейся в БД, может быть обеспечен простыми средствами, не требующими передачи по сети больших объемов информации, что и является главным преимуществом такого типа архитектуры.

Рассмотрим основные средства Delphi для реализации трехзвенной архитектуры.

Создание сервера приложений

На первом шаге необходимо создать сервер приложений. Для этого нужно создать новый проект. В меню **File** нужно выбрать опцию **New**/**Other**, в открывшемся окне перейти на вкладку **Multititer** и выбрать компонент **Remote Data Module**. В появившемся окне в строке **CoClass Name** следует ввести название класса **Server**. После этого в проекте появится новый модуль с соответствующим именем. Назначение этого компонента заключается в обеспечении связи между источником данных и клиентом.

Теперь поместим в это окно компонент **DataBase** с вкладки **BDE** палитры компонентов. Для связи этого компонента с БД в свойстве **AliasName** укажем псевдоним **Mydb**. В свойство **DataBaseName** нужно ввести значение **serv**.

Теперь в это же окно поместим компонент **Table,** и в его свойстве **DataBaseName** сошлемся на соответствующее свойство компонента **DataBase**. Затем, изменяя свойство **TableName**, свяжем компонент **Table**, например, с таблицей **Otdel**. Если все сделано

правильно, то окно дерева объектов должно иметь такой вид, как на рис. п.32.

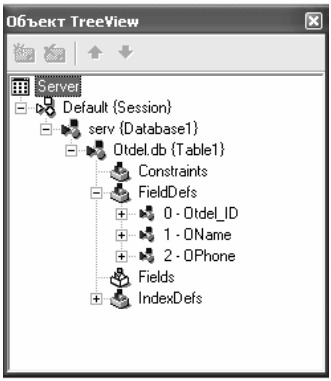


Рис. п.32. Окно дерева объектов

Далее в окно **Server** поместим компонент **DataSetProvider** с вкладки **DataAccess**. В его свойстве **DataSet** сошлемся на таблицу **Table1**. Нужно сохранить созданный проект под именем **MyServer** в новой папке с таким же названием. На последнем шаге запустим проект на исполнение на 1-2 секунды для того, чтобы вновь созданный сервер был зарегистрирован в системном реестре **Windows**.

Создание клиента

Для создания клиента необходимо создать новый проект в Delphi и поместить на главную форму компоненты **SocketConnection** с вкладки **DataSnap**, **ClientDataSet** с вкладки **DataAccess**, а также компоненты **DataSource**, **DBGrid** и **Button**.

Компонент **SocketConnection** предназначен для осуществления связи между клиентом и сервером приложений со стороны клиента и использует для этого специальные функции Windows, которые называются сокетами. Для работы с сокетами Delphi имеет утилиту **scktsrvr.exe**, которая находится в папке **Bin** каталога Delphi. Нужно запустить эту утилиту на исполнение, после чего в правом углу панели инструментов рабочего стола появится значок **Borland Socket Server**.

Компонент **SocketConnection** имеет свойство **host**, в котором указывается имя компьютера, на котором размещен сервер приложений. Поскольку в данном случае как клиент, так и сервер приложений располагаются на одном и том же компьютере, в это свойство следует установить значение **localhost**.

В свойстве **ServerName** следует сослаться на созданный сервер **MyServer**.**Server**, а в свойстве **Connected** установить значение **true**. В этот момент на экране появится новое окно с названием

Form1 – это автоматически запустился созданный сервер приложений. Его можно свернуть, но не завершить работу – иначе клиент не сможет установить связь с сервером приложений.

Теперь нужно связать компонент **ClientDataSet** с компонентом **SocketConnection**, сославшись на него в свойстве **RemoteServer**, а в качестве имени провайдера в поле **ProviderName** выбрать **DataSetProvider1** – это тот самый компонент, который был добавлен в последнюю очередь в проект сервера приложений.

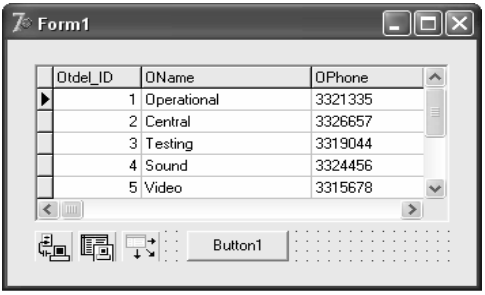


Рис. п.33. Окно программы клиента

Далее свяжем сетку **DBGrid** с компонентом **DataSource,** а компонент **DataSource** – с компонентом **ClientDataSet**. После этого в окне **DBGrid** должно отобразиться содержимое полей таблицы **Otdel** (рис. п.33). Если этого не произошло, то прежде всего необходимо проверить значение свойства **Active** компонента **ClientDataSet**.

Клиентский проект можно сохранить в новую папку **MyClient** под тем же названием.

Теперь перейдем к программированию кнопки **Button1**. Она создана для того, чтобы клиент мог вносить изменения в таблицу **Otdel** и таким образом менять содержимое БД. Для этого следует обработчик события **OnClick** запрограммировать следующим образом:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with ClientDataSet1 do
ApplyUpdates(0);
end;
```

Если теперь запустить проект на исполнение и, изменив содержание какой-либо записи, нажать на эту кнопку, то вызов метода **ApplyUpdates** приведет к тому, что сделанное изменение перепишется в БД. Если добавить в проект компонент **DBNavigator**, то с его помощью можно будет также удалять и добавлять записи в БД.

Сергей Львович Шнырев

БАЗЫ ДАННЫХ

Учебное пособие

Редактор Т.В. Волвенкова

Подписано в печать 15.12.2010. Формат 60×84 1/16.
Печ. л. 14. Уч.-изд. л. 14. Тираж 100 экз.
Изд. № 1/4/9. Заказ № 21

Национальный исследовательский ядерный университет «МИФИ»
115409, Москва, Каширское ш., д. 31

ООО «Полиграфический комплекс «Курчатовский».
144000, Московская область, г. Электросталь, ул. Красная, д. 42