М.В. Гранков, А.И. Жуков

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ СУБД ПРИ РАЗРАБОТКЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Ростов-на-Дону
2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

М.В. Гранков, А.И. Жуков

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ СУБД ПРИ РАЗРАБОТКЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Ростов-на-Дону
2013

УДК 004.65
Г 75

Рецензент
доктор технических наук, профессор *Р.А. Нейдорф*

Гранков М.В.
Г 75 Использование реляционных СУБД при разработке информационных систем: учеб. пособие / М.В. Гранков, А.И. Жуков. – Ростов н/Д: Издательский центр ДГТУ, 2013. – 99 с.

ISBN 978-5-7890-0852-2

В данном пособии представлены основы проектирования реляционных баз данных, а также рассмотрены методы и средства разработки программных модулей для использования реляционных баз данных в информационных системах различного назначения. Учебное пособие предназначено для студентов, обучающихся на специальностях 230105 – Программное обеспечение вычислительной техники и автоматизированных систем, 010305 – Математическое обеспечение и администрирование информационных систем, 231000 – Программная инженерия, 090301 – Компьютерная безопасность, 231300 – Прикладная математика и изучающих курсы «Базы данных», «Системы управления базами данных» и «Современные СУБД».

Пособие соответствует программе дисциплины и общеобразовательным стандартам и снабжено вопросами и заданиями для самоконтроля. Данное пособие может быть использовано при написании лабораторных, практических, курсовых и дипломных работ.

УДК 004.65

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Научный редактор
доктор технических наук, профессор В.Г. Кобак

ISBN 978-5-7890-0852-2

Предисловие

Современный мир находится на пути перехода к информационному обществу, в котором существенна роль информационных технологий, основой которых является информация в любых ее формах. Трудно сказать, что можно считать началом этого процесса, однако, известно, что впервые термин «информационное общество» появился в 60-х годах XX века и был обоснован, прежде всего, прогрессом информационных технологий. В это же время впервые появился и термин «база данных», который был связан с появлением компьютерных систем и программного обеспечения обработки записей, сохраняемых на перфокартах. В дальнейшем, на протяжении более чем полувековой истории, роль баз данных и программного обеспечения, используемого для управления и взаимодействия с ними, постоянно росла.

Начало XXI века ознаменовано стремительным ростом популярности Интернета, развитием аппаратного обеспечения, позволившим увеличить на порядки объем запоминающих устройств и скорость обмена информацией. Объем баз данных существенно возрос, а проблема увеличения скорости доступа (добавления, модификации, поиска) к большим массивам информации остается актуальной и по сей день.

В настоящее время наиболее распространенными являются базы данных, основанные на реляционной модели, теория которых был разработана еще в 1970-х годах. Именно поэтому в данном учебном пособии основное внимание уделяется реляционным системам управления базами данных, а именно: принципам их организации, вопросам проектирования баз данных и взаимодействия с ними по средствам языка SQL, а также разработке прикладных программ.

Пособие предназначено для студентов высших учебных заведений, которые впервые знакомятся с принципами использования реляционных баз данных при разработке программных средств.

1. ВВЕДЕНИЕ В ТЕОРИЮ БАЗ ДАННЫХ

1.1. Основные понятия и определения

1.1.1. Информационные системы и базы данных

В соответствии с законодательством РФ информационной системой (ИС) является организационно упорядоченная, в том числе с использованием средств вычислительной техники и связи, реализующих информационные процессы, совокупность документов (массивов документов) и информационных технологий [13]. В рамках данного пособия будем рассматривать, прежде всего, автоматизированные информационные системы (АИС), основанные на использовании аппаратных и программных средств для хранения, обработки, поиска и предоставления информации [13, 15].

Основным ресурсом и одновременно объектом управления ИС является информация. До сих пор не существует точного определения понятия «информация», а его использование в различных областях знания отличается специфическим набором свойств. В рамках теории баз данных (БД) принято разделять ***данные***, представляющие (декларирующие) некоторые факты, и ***информацию*** – данные, обработанные специальным образом [8, 18]. Основное отличие информации от данных в том, что первая имеет конкретное предназначение, тогда как данные лишь определяет параметры объекта в некоторый момент времени в рассматриваемой предметной области. Очевидно, что граница между этими понятиями не всегда может быть четко определена, поэтому ряд авторов [8, 13] при описании функционирования и принципов организации БД рассматривают данные понятия как синонимичные.

Хранение данных в БД и автоматизированная обработка данных для получения информации подразумевает использование программно-аппаратного комплекса, входящего в состав АИС. База данных, таким образом, является составным элементом АИС, и в широком смысле представляет совокупность взаимосвязанных единиц данных [**18]**. В узком смысле, ***база данных*** или ***система баз данных*** – это компьютеризированная система, представляющая собой специальным образом структурированную совокупность данных, для хранения которых используются разнородные запоминающие устройства, а основным назначением является хранение данных с возможностью их извлечения и модификации [8, 20]. В соответствии с зако-

нодательством РФ, «база данных – объективная форма представления и организации совокупности данных, систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ» [13].

Следующие аспекты отражают преимущества от использования систем баз данных в качестве средства хранения данных, по сравнению с традиционными «бумажными» методами хранения и обработки данных [8]:

- компактность – физический объем (в пространстве) цифровых носителей несоизмеримо меньше объема бумажных источников (журналы, книги и т.д.);
- скорость – поиск и модификация данных в ЭВМ при больших объемах данных на порядки быстрее;
- низкие трудозатраты – вычислительные машины лучше выполняют монотонную и однообразную работу, чем люди;
- актуальность – возможность в любой момент быстро получить самую последнюю информацию о состоянии хранимых объектов.

Базы данных принято классифицировать по различным характеристикам:

- по способу обработки данных:
  - централизованные – физически расположена в памяти одной ЭВМ, при этом возможен распределенный доступ к БД с использованием аппаратного и программного обеспечения вычислительной сети;
  - распределенные – состоят из нескольких пересекающих, а возможно, даже дублирующих друг друга частей, расположенных в памяти нескольких ЭВМ, функционирующих в рамках одной вычислительной сети;
- по способу доступа к данным:
  - БД с локальным доступом к данным;
  - БД с удаленным (сетевым) доступом к данным, которые в свою очередь делятся на:
    - БД с файл-серверной архитектурой;
    - БД с клиент-серверной архитектурой.

Таким образом, можно выделить как минимум две важнейшие характеристики баз данных:

- во-первых, база данных является формализованной на некотором языке моделью предметной области из реального мира;

- во-вторых, база данных может лишь хранить и предоставлять факты, которые сами по себе не имеют большого практического значения для пользователей, т.е. требуется специальный комплекс программного обеспечения, реализующий преобразование данных в информацию.

1.1.2. Уровни абстракции в базах данных

Как было отмечено выше, база данных представляет некоторую информационную модель предметной области. Кроме того, система баз данных предназначена для хранения и предоставления по требованию пользователей данных об объектах реального мира. При этом очевидно, что в БД необходимо хранить данные лишь о тех объектах и их свойствах из предметной области, которые необходимы для пользователей, использующих систему баз данных. Например, в базе данных мебельного салона для описания объекта «Мебель» с точки зрения менеджера продаж требуется хранить реквизиты: габариты (длина × ширина × высота), материал, цвет. При этом с точки зрения службы доставки важно также знать примерный вес изделия, зато к параметру «цвет» грузчики в относятся индифферентно.

Этот умозрительный пример демонстрирует различия восприятия одной предметной области различными пользователями базы данных. При этом часто ни один из пользователей при составлении описания не может охватить всех объектов и их реквизитов, существующих в реальном мире. Данный факт иллюстрируется ниже (рис.1). Объекты (или элементы данных), представленные кругами, могут принадлежать одному или нескольким представлениям пользователей.

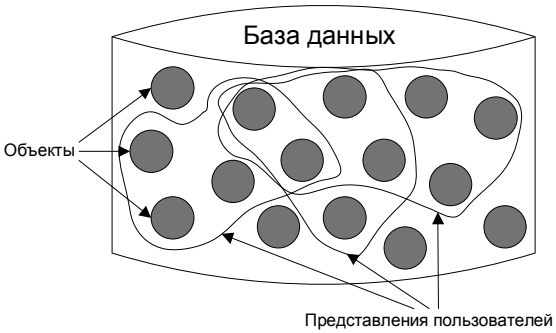


Рис.1. Представления пользователей об объектах реального мира

Таким образом, база данных представляет собой совокупность всех описаний ее пользователей, среди которых выделяются основные объекты и свойства и отбрасываются второстепенные. Этот процесс называется абстрагированием и является основополагающим при проектировании БД.

В процессе эксплуатации баз данных часто возникают ситуации, когда перед программными средствами, использующими ее, ставятся новые требования, либо условия в предметной области существенно меняются. Причины формирования новых требований к программному обеспечению могут быть обусловлены появлением новых законодательных актов, изменением бизнес-процессов, связанным с расширением сфер деятельности предприятия и т.д. Изменение требований к программному обеспечению может, в свою очередь, вызвать изменение требований к базе данных: потребуется добавление новых объектов и их свойств либо модификации структуры уже хранящихся в БД объектов.

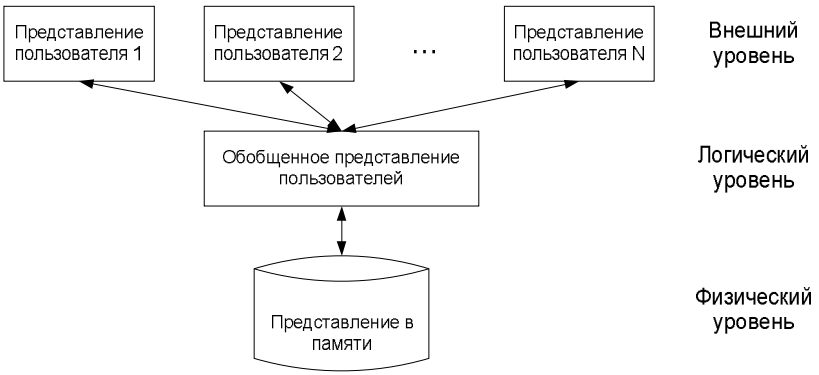


Рис.2. Трехуровневая схема организации БД

Таким образом, необходима такая архитектура базы данных, которая позволила бы весь период эксплуатации БД обеспечивать стабильное развитие и, при необходимости, простую модификацию БД. В качестве такой архитектуры комитетом ANSI/SPARC была предложена трехуровневая система организации данных [2, 13]. Эта архитектура состоит из трех уровней абстракции (рис.2):

1) внешняя (концептуальная) модель или уровень представления пользователей, на котором предметная область описывается с точки зрения конечных пользователей программного средства, т.е. таких представлений может быть несколько;

2) логический уровень, который объединяет в себе представления пользователей и представляет одно обобщенное описание предметной области;

3) внутренний (физический) уровень, который реализует описание даталогической модели на некотором формальном языке, т.е. представляет физическую реализацию базы данных в конкретной СУБД.

При проектировании и создании базы данных процесс перехода от объектов реального мира к их информационной модели заключается в последовательном переходе от уровня представлений пользователей к физической реализации БД. Рассмотрим программный инструментарий, используемый для работы с БД.

1.1.3. Системы управление базами данных

Система управления базами данных (СУБД) – комплекс языковых, математических и программных средств, предназначенный для создания, поддержки, модификации и совместного использования данных, хранящихся в базе данных, одним или несколькими лицами [8, 13, 20]. Основным назначением СУБД является [8, 13]:

- обеспечение пользователя инструментом, позволяющим оперировать данными в терминах, не связанных с особенностями их хранения в ЭВМ;
- разграничение прав доступа к информации;
- поддержка целостности и непротиворечивости данных в БД;
- синхронизация доступа к информации при одновременном обращении нескольких пользователей;
- восстановления состояния базы данных после различных категорий отказов.

Независимость данных является основным качественным отличием систем управления базами данных от предшествующих автоматизированных систем обработки информации. Для предшествую-

щих систем (например, для систем, в которых хранение информации основано на типизированных файлах, хранящих записи определенного формата) информация о структуре данных и способах получения доступа к ним были встроены в исполняемый код приложения. В связи с этим в случае изменения требований к БД (например, в случае необходимости увеличить размер поля) приходилось менять структуру записей в файле, а также изменять код программных модулей. СУБД построены таким образом, чтобы обеспечивать развитие БД, не оказывая существенного влияния на уже разработанные прикладные программы.

Три уровня абстракции в архитектуре БД (внешний – логический – внутренний) обеспечивают *два уровня независимости данных* в СУБД [13]:

- первый уровень обеспечивает отображение «логический-внутренний» и называется *физической независимостью данных*, при этом, так как правила представления логических записей на внутреннем уровне описываются в отдельном файле, то при изменении внутренней структуры БД изменяется и отображение «логический-внутренний» таким образом, чтобы логическая схема осталась неизменной;
- второй уровень обеспечивает отображение «внешний-логический» и называется *логической независимостью данных*, которая определяет соответствие между некоторым концептуальным представлением пользователей БД и логической схемой.

Приведем два примера, демонстрирующих преимущества принципа независимости данных. Предположим, что некоторое приложение работает с хранимым файлом, в котором описаны записи, имеющие два поля «А» и «Б». Физическая независимость данных подразумевает, что в случае замены хранимого файла на файл, в котором поля «А» и «Б» будут поменяны местами (при этом соответствующим образом будет изменено отображение «логический-внутренний»), работоспособность прикладных программ от этого не пострадает. Другой пример: в процессе функционирования БД потребовалось поддерживать новое свойство объектов, что было определено одним из пользователей системы. В результате изменилось внешнее представление без необходимости модификации уже существующих приложений. Очевидно, что в случае некоторых модификаций логической схемы потребуется изменение кода функциони-

рующих приложений. В первую очередь речь идет об удалении данных об объектах и/или их свойствах из БД.

Таким образом, функционирование СУБД может быть описано последовательностью следующих этапов:

1) пользователь формирует запрос на доступ или модификацию данных;

2) СУБД перехватывает и анализирует запрос;

3) СУБД строит преобразование «внутренний-логический» и «логический-внешний»;

4) СУБД выполняет необходимые операции над хранимой БД.

Еще одним важным понятием, связанным с функционированием СУБД, является целостность. Под *целостностью базы данных* понимают свойство БД, при наличии которого БД содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений. При этом *ограничения целостности* – это набор условий, определяющих целостное состояние БД, т.е. только в случае их выполнения для текущего состояния БД можно говорить о том, что БД содержит полные и непротиворечивые данные. Ограничения целостности определяются в соответствии с особенностями, условиями и ограничениями, имеющими место в предметной области. Например, если предположить, что номер зачетной книжки является уникальным для любого студента в вузе, то появление в БД двух записей о разных студентах с одним номером зачетной книжки является нарушением целостности. Поддержание БД в целостном состоянии должно автоматически выполняться СУБД.

СУБД должна выполнять следующие *функции:*

- допускать определение хранимых данных (концептуальная, логическая, физическая схемы);
- обрабатывать запросы пользователей на выборку, изменение, удаление существующих данных в БД или на добавление новых данных;
- контролировать пользовательские запросы и пресекать попытки нарушения целостности данных или несанкционированного доступа;
- осуществлять контроль за восстановлением данных после отказа и за дублированием данных;
- обеспечить функцию *словаря данных*, т.е. специализированной области БД, в которой хранится метаданные (данные о данных), определяющие другие объекты системы баз данных;
- обеспечить выполнение всех функций с максимально возможной производительностью.

1.1.4. Компоненты системы баз данных

Упрощенная схема функционирования системы баз данных, определяющая ее основные компоненты: данные, аппаратное обеспечении, программное обеспечение и пользователи [8], демонстрируется на рис.3.

Данные хранятся в БД в некотором структурированном виде, который зависит от выбранной модели данных (см. раздел 1.2). В качестве элементов структуры БД, используемых для хранения информации, могут выступать таблицы, поля, узлы графа, списки, xml-документы и т.д.

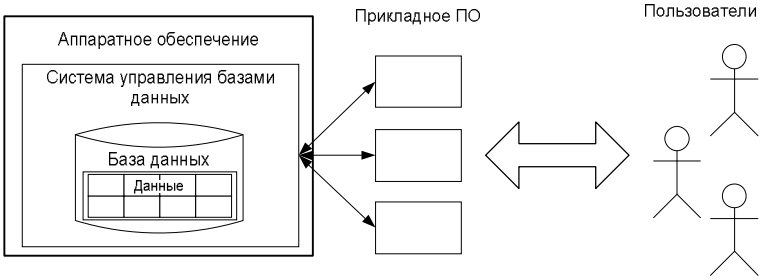


Рис.3. Компоненты системы баз данных

Абстрагируясь от структур данных, используемых СУБД, можно констатировать, что в итоге все данные, хранимые в БД, сохраняются в памяти ЭВМ в виде файлов. В первой главе без ориентации на конкретную модель данных или СУБД будем использовать понятие «файл данных» как элементарную структурную единицу. Файл данных состоит из структурированных особым образом элементов данных, описывающих объекты предметной области. Совокупность таких файлов данных (*хранимых файлов*) составляет базу данных.

Перед рассмотрением свойств данных в БД требуется определить понятие «избыточность данных». Опишем условия возникновения избыточности данных: в составе АИС имеются несколько приложений, каждое из которых работает с одним или несколькими файлами данных. При этом данные в этих файлах частично или полностью дублируют друг друга. Например, имеется файл данных с информацией о студентах – STUDENT (ФИО, номер зачетной книжки, Специальность) и с информацией о составе учебных групп – GROUP (Название группы, ФИО студента, Специальность). Информация о

том, какой специальности обучается студент, хранится в двух файлах; таким образом, данные в файлах частично перекрываются, что потенциально может привести к появлению проблем. Например, если студент переведется с одной специальности на другую, то потребуется модификация записей в нескольких файлах данных, если при этом эти файлы используются независимо друг от друга разными приложениями – возможно нарушение целостности.

Таким образом, об избыточности данных можем говорить в случае наличия в системе общих данных, копии которых одновременно хранятся (дублируются) в различных файлах данных [8, 18].

Данные в БД обладают двумя важнейшими свойствами, обуславливающими преимущества использования СУБД по сравнению с применением файловых систем для хранения больших массивов данных: интегрированность и разделяемость [8]. Наилучшим образом демонстрация принципов, заложенных в основе этих свойств, представляется примерами. Допустим, в БД имеется файл с записями о студентах – STUDENT, в котором хранится информация о его фамилии, имени, отчестве и номере зачетной книжки. Также в БД имеется файл SUBJECT, в котором хранится название изученной студентом дисциплины, оценка и номер зачетки студента. Очевидно, не имеет смысла включать информацию из SUBJECT в файл STUDENT, поскольку такая информация всегда может быть получена для каждого студента из файла SUBJECT (по номеру зачетной книжки). Кроме того, такое объединение, очевидно, приводит к возникновению избыточности.

Этот пример демонстрирует свойство *интегрированности данных:* возможность представить базу данных как объединение нескольких отдельных файлов данных. С другой стороны, информация о каждом студенте может быть одновременно доступна сотрудникам деканатов, учебного отдела или отдела по работе с обучающимся. Возможно также, что информацию о текущей успеваемости доступна через web-приложения родителям студента. В данном случае, речь идет о *разделяемости данных*, т.е. о возможности использования данных в БД несколькими различными пользователями параллельно.

Все данные, которые хранятся в БД и используются прикладными программами, называются *постоянными* по отношению к *временным* данным (входные, промежуточные, выходные и т.д.).

Функционирование любой АИС сопряжено с необходимостью использования соответствующего программного и аппаратного обес-

печения и системы баз данных не являются исключением. Для управления БД применяются СУБД, а также дополнительные утилиты, обеспечивающие копирование данных, защиту и восстановление после отказов, выполнение запросов к БД, средства автоматизированной разработки приложений, средства проектирования, генераторы отчетов и др. Для функционирования СУБД и другого программного обеспечения требуются: ЭВМ, включающие в свой состав накопители жестких дисков, оперативную память, процессор, а также различное сетевое оборудование.

Последним (по порядку, но не по значению) компонентом систем баз данных являются пользователи, которых принято делить на три группы [8, 13]:

- прикладные программисты, отвечающие за написание прикладных программных средств, использующих БД;
- конечные пользователи – люди из предметной области, автоматизация труда которых является целью использования системы баз данных и которые работают с ней, по средствам применения прикладных программ, разработанных программистами;
- администратор базы данных – «лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение» [20].

Администратор БД – это важнейший пользователь БД, который несет ответственность за работоспособность системы. Именно от квалификации этого пользователя в существенной степени зависит производительность не только системы БД, но также и прикладных программных средств ее использующих. Приведем список функций администратора БД:

- помощь в создании схем внешнего уровня (представлений пользователей);
- определение схемы логического уровня;
- создание внутренней схемы (физическое представление БД);
- взаимодействия с конечными пользователями (предоставление полномочий на доступ к данным) и прикладными программистами (консультации по разработке программного обеспечения);
- определение правил безопасности и целостности;
- определение процедур резервного копирования данных с целью защиты от отказов системы [8];
- контроль производительности системы и реагирование на изменяющиеся требования к компонентам БД.

1.2. Модели данных

Американский математик Э. Кодд определил модель данных как комбинацию трех компонентов, устанавливаемых в отношении любой БД, соответствующей модели [2, 11]:

- множество типов объектов данных, образующих базовые элементы для определения БД;
- множество общих правил целостности, ограничивающих набор экземпляров тех типов объектов, которые могут появляться в БД;
- множество операций, применимых к экземплярам объектов, хранимых в БД для выборки и других целей.

Таким образом, *модель данных* представляет собой формальную теорию, определяющую способ представления, структурирования, манипулирования данными, а также методы описания и поддержки целостности БД.

На протяжении полувекового развития теории баз данных были выделены следующие модели данных (первые три являются классическими, остальные получили развитие в последние два десятилетия) [20]:

1. *Иерархическая модель* предполагает, что между элементами данных возникают связи, которые описываются с использованием структуры данных упорядоченного графа (или дерево). Тип «дерево» может содержать «поддеревья», т.е. является составным. Каждый из типов «дерево» в обязательном порядке имеет один «корневой» тип и упорядоченное множество (возможно пустое) подчиненных типов. Иерархическую модель данных использовали первые СУБД, однако, впоследствии ее популярность значительно снизилась с распространением реляционной модели. На базе данной модели в начале 1990-х годов были разработаны БД, основанные на файлах XML и послужившие основой для определения в начале XXI века нового направления развития систем баз данных – документно-ориентированных СУБД.

2. *Сетевая модель* основана на графовых структурных данных. Для описания используются два основных понятия: «узел» (иногда запись) и «связь». Считается наиболее удобной в случае, если требуется представить множество объектов и связей между ними. В отличие от иерархической модели, каждый узел может иметь не только нескольких детей, но и несколько родителей. Использование сетевых

моделей при разработке БД было впервые предложено в 1980-х годах, однако, большой популярности СУБД, основанные на данной модели, долгое время не получали в связи со сложностью их организации. Начиная с 2007-го года на базе сетевой модели активно разрабатываются и используются графовые (иногда графо-ориентированные) системы баз данных, которые, правда, отличаются от сетевых наличием возможности хранения графов любого вида [5].

3. *Реляционная модель* была предложена сотрудником фирмы IBM Э. Коддом и основана на теоретико-множественном понятии «отношение». Так как модель является наиболее популярной по количеству СУБД, функционирующих в различных проектах и основанных на базе данной модели, в связи с этим в данном пособии отводится целая глава (вторая) на описание особенностей реляционной модели и принципов разработки реляционных баз данных.

4. *Постреляционная модель*. Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц, однако, существует ряд случаев, когда это ограничение снижает эффективности реализации прикладных программ. Постреляционная модель является расширением традиционной реляционной модели в плане снятия указанного ограничения. Кроме того, постреляционная модель поддерживает ассоциированные многозначные поля, совокупность которых представляет ассоциации. В рамках одной ассоциации значения одного или нескольких полей записей являются идентичными для всех записей, включенных в ассоциацию [20]. В целом, можно сказать, что постреляционная модель решает некоторые проблемы реляционной модели, однако, большого распространения данная модель не получила.

5. *Объектно-ориентированная модель* основана на объектно-ориентированном подходе к организации данных. Наиболее важным качественным аспектом объектно-ориентированных систем баз данных является возможность определения поведенческого аспекта данных в БД, представленных в виде объектов. Данный тип БД возник еще в середине 80-х и в 90-х ему были выданы большие авансы как средству улучшения реляционной технологии, однако, на сегодняшний день объектно-ориентированные БД серьезно конкурировать с реляционными БД не могут.

6. *Объектно-реляционная модель* была получена на основе реляционной с добавлением некоторых возможностей объектно-ориентированной модели: идентификация каждой записи (объекта),

наличие классов и наследования, возможность создания хранимых процедур, а также возможность описания пользовательских типов данных. Некоторые реляционные СУБД в процессе эволюции приобрели объектные возможности (PostgreSQL, Oracle и др.).

Перечисленные выше преимущества и недостатки различных моделей данных представлены в табл.1.

Таблица 1

Достоинства и недостатки известных моделей данных

Название модели (известные СУБД)	Достоинства	Недостатки
1	2	3
Иерархическая (классические: IMS, ИНЭС, ДОСУБД: MongoDB, CouchDB, Lotus Notes и др.)	Эффективное использование памяти; относительно быстрая скорость выполнения операций над данными; удобна для работы с иерархически упорядоченной информацией	Громоздкость для обработки информации со сложными логическими связями; сложность понимания обычным пользователем
Сетевая (классические: IDMS, КОМПАС, графовые: OrientDB, Neo4j и др.)	Эффективная реализация по показателям затрат памяти и оперативности; возможности по организации произвольных связей	Высокая сложность и жесткость схемы БД; ослаблен контроль целостности связей (вследствие допустимости любых связей)
Реляционная (Fox Pro, MySQL, MS Access, Oracle, MS SQL Server, PostgreSQL, SQL Lite и др.)	Простота, понятность за счет близости с предметной областью, а также удобство физической реализации. Хорошопроработанная формальная теория определения схемы БД	Сложность описания иерархических и сетевых связей, проблема хранения записей переменной длины
Постреляционная (uniVerse, Dasbd)	Возможность представления совокупностей связанных реляционных таблиц одной таблицей, что обеспечивает повышение эффективности обработки информации и ее наглядность	Сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных

Окончание табл.1

1	2	3
Объектно-ориентированная (Cache, Orion, Jasmine)	Возможность отображения информации о сложных взаимосвязях объектов; возможность идентификации отдельной записи БД и ее поведения (методы)	Высокая понятийная сложность; неудобство обработки данных; проблема выполнения произвольных запросов к данным; низкая скорость выполнения запросов
Объектно-реляционная (Oracle, PostgreSQL версии 8 и старше)	Расширение реляционной модели с добавлением объектных возможностей	Те же, что и у реляционных систем; эффективность применения объектных технологий зависит от их реализации в конкретной СУБД

Необходимо отметить, что модели данных определяют правила и язык для выполнения моделирования (т.е. представляют инструмент моделирования), а результатом моделирования является конкретная схема базы данных. *Схема базы данных* «включает в себя описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных» [7].

1.3. Основные этапы проектирования БД

Как уже было отмечено выше, независимо от выбранной модели данных процесс проектирования проходит в три этапа:

- выполнение концептуального проектирования внешнего уровня;
- обобщение представлений пользователей и выполнение логического проектирования;
- реализация схемы базы данных на внутреннем (физическом) уровне.

Рассмотрим эти этапы подробнее, а также определим инструментарий выполнения проектирования на различных уровнях абстракции.

1.3.1. Концептуальное проектирование

Концептуальное (инфологическое) проектирование заключается в построении семантической модели предметной области [13, 14], т.е. модели предназначенной для смыслового описания объектов предметной области на самом высоком уровне абстракции:

без использования понятий, связанных со спецификой физической реализации БД. Таким образом, модель концептуального уровня создается без опоры на какую-либо модель данных или конкретную СУБД. Внешний уровень описывается совокупностью представлений различных пользователей и использует понятия из предметной области.

Как правило, концептуальное проектирование проводится разработчиком БД с участием конечных пользователей. На первом этапе происходит опрос пользователей с фиксацией основных информационных объектов и их свойств или понятий предметной области, а также зависимостей, возникающих между ними. Затем описываются ограничения целостности, т.е. требования к допустимым значениям для различных информационных объектов системы.

В качестве формального аппарата представления инфологической модели обычно используются ER-диаграммы (англ. entity-relationship), иначе, диаграмма «сущность-связь». Данная модель относится к семантическим моделям, вследствие чего имеет место тесная логическая связь с предметной областью. Основными понятиями являются «объект» («сущность») и «связь».

Объект – семантическое понятие, которое может быть полезно при обсуждении устройства реального мира, т.е. нечто существенное, полезное для описания предметной области при этом объекты не обязательно должны быть материальными. Каждый объект обладает рядом свойств, которые также называют *атрибутами* или *реквизитами* объекта. Различают понятия *тип атрибута* и *значение атрибута*. *Тип объекта* состоит из упорядоченной последовательности *типов атрибутов* и определяет набор свойств, которые содержат объекты данного типа.

Между объектами возникают связи различных видов. Рассмотрим суть связей различных видов с использованием двух условных типов объектов из предметной области A и B. Между экземплярами A и B могут возникать связи следующих типов:

- один-к-одному (1:1) – каждому объекту типа A может быть поставлено в соответствие не более одного объекта типа B, при этом обратное также справедливо (например, один человек может иметь только один паспорт РФ и один паспорт РФ принадлежит всегда только одному человеку);
- один-ко-многим (1:N) – каждому объекту типа A может быть поставлено в соответствие любое число объектов типа B, при

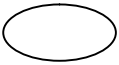
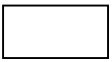
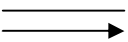
этом каждому объекту типа B соответствует не более одного объекта типа A (например, один человек может иметь несколько документов о получении высшего образования, но каждый документ о получении высшего образования относится только к одному человеку);

- многие-к-одному (N:1) – обратный вид связи относительно связи «один-ко-многим»;
- многие-ко-многим (N:N) – каждому объекту типа A может быть поставлено в соответствие любое число объектов типа B, при этом обратное также справедливо (например, один человек может работать в нескольких фирмах и при этом в каждой фирме может работать несколько человек).

Обозначения, принятые при изображении ER-диаграмм представлены в табл.2.

Таблица 2

Обозначения элементов ER-диаграммы

Блок	Описание
	Атрибут объекта, название указывается внутри
	Сущность (объект)
	Связь (отношение)
	Соединительные линии. Используются для связи атрибутов с объектами и связями, а также для обозначения отношений, возникающих между различными объектами. Стрелка используется для указания направления, если необходимо.

На диаграмме (рис. 4) представлены объекты, существующие в предметной области (Человек, Студент, Группа, Кафедра, Направление, Профиль, Приказ, Студенческий план, Оценка, Учебный план, Дисциплина), их атрибуты: (например, для объекта Человек определены атрибуты фамилия, имя, отчество, дата рождения и пол), а также связи, возникающие между объектами (например, «Студент есть Человек»). Так как один человек может одновременно обучаться на нескольких специальностях, т.е. быть одновременно несколькими студентами, то между объектами Человек и Студент имеет место связь один-ко-многим.

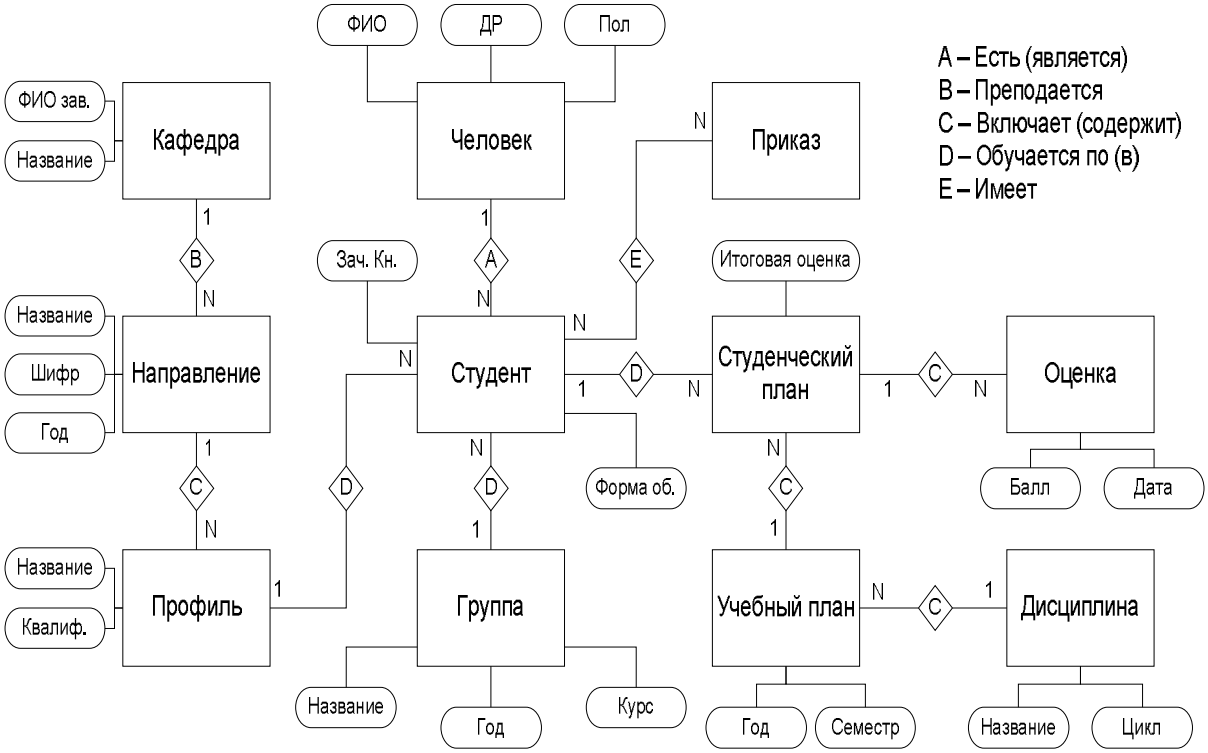


Рис.4. ER-диаграммы

Основными преимуществами ER-моделей являются [14]:

- наглядность;
- возможность проектирования БД с большим числом объектов и атрибутов;
- имеются реализации ER-моделей во многих системах автоматизированного проектирования БД (например, ERWin и др.).

1.3.2. Логическое проектирование

Логическое (даталогическое) проектирование заключается в создании схемы БД на базе концептуальных моделей (может быть одна модель) путем формализации понятий из предметной области и их представления в терминах выбранной модели данных.

Если концептуальное проектирование выполнялось в некоторой автоматизированной среде, то построение логического проекта БД может быть в существенной степени автоматизировано. Пример даталогической модели для реляционной БД, на которой указаны таблицы, имена полей, а также первичные и внешние ключи, представлен на рис.5.

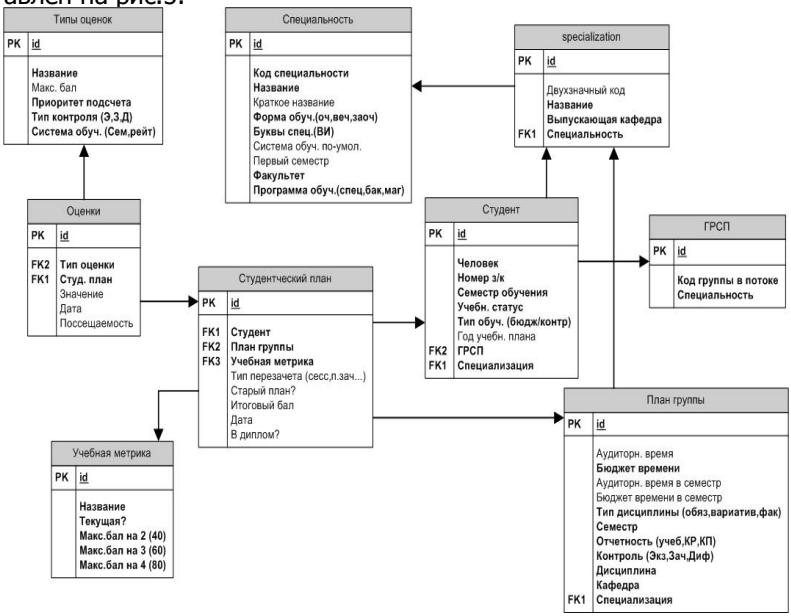


Рис.5. Пример даталогической модели для РБД

1.3.3. Физическое проектирование

Заключительный этап проектирования состоит в создании схемы базы данных для конкретной СУБД. Такая схема, представляющая структуру БД и описанная на формальном языке, хранится в словаре данных СУБД. *Словарь данных* или *словарь вычислений* – это «центральное хранилище информации о данных, такой как значение, взаимосвязи с другими данными, их источник, применение и формат» [1]. Другими словами, словарь данных СУБД содержит метаданные – информацию об объектах СУБД. Словарь данных часто называют *системным каталогом*.

В зависимости от СУБД физический проект может включать в себя ограничения на именования объектов базы данных, различные ограничения целостности (уникальное значение, внешний ключ, диапазон допустимых значений и т.п.), ограничения на поддерживаемые типы данных и т.д. Кроме того, специфика конкретной СУБД влияет на различные физические аспекты представления данных в памяти ЭВМ [14]: выбор методов управления дисковой памятью, разделение базы данных по файлам и устройствам хранения, выбор методов доступа к данным, создание и структуру индексов и т.д.

1.4. Использование базы данных в многопользовательском режиме

Подавляющее большинство систем БД поддерживают многопользовательский доступ. Требования к многопользовательским БД заключаются в том, чтобы действия с данными в них со стороны пользователя не отличались от того, как эти действия выполнялись бы в однопользовательской системе. В данном параграфе рассматриваются некоторые методологии, применяемые для обеспечения многопользовательского доступа к данным в БД.

1.4.1. Транзакции

Транзакция – это атомарная логическая единица работы, выполняемой СУБД над данными, хранящимися в БД. Каждое слово в этом определении не случайно. Во-первых, несмотря на то, что одна транзакция может объединять в себе большое число команд, транзакция является неделимой, т.е. либо успешно выполняются все ко-

манды, входящие в транзакцию, либо ни одна из них. Во-вторых, объединение команд в транзакцию обуславливается рассуждениями, в основе которых лежит семантика взаимодействия объектов в предметной области и бизнес-логика функционирования прикладных программ. Наконец, выполнение транзакций происходит в рамках СУБД, при этом целью выполнение транзакции является доступ к данным с целью выборки, модификации, добавления или удаления.

Система баз данных, поддерживающая обработку транзакций, гарантирует, что если в ходе выполнения неких обновлений данных в рамках одной транзакции произойдет ошибка (например, нарушение целостности данных или системный сбой), то все эти обновления будут отменены [8]. Любая транзакция может завершиться одним из событий: фиксация транзакции (в случае успешного выполнения каждой команды составляющей транзакцию) или откатом БД на момент начала транзакции. Для фиксации транзакции применяется оператор COMMIT, для отката – оператор ROLLBACK.

СУБД гарантирует, что в случае фиксации транзакции БД будет находиться в непротиворечивом состоянии, т.е. целостность данных нарушена не будет. При этом контроль целостности данных в ходе выполнения транзакции не выполняется. Невозможно предположить, что все прикладные программы будут взаимодействовать с СУБД, контролируя все типы ошибок, возникающих в процессе этого взаимодействия. Поэтому если транзакция не была зафиксирована (например, из-за того, что программа не была нормально завершена, не было закрыто соединение с БД, не был передан оператор COMMIT), то система БД автоматически выполняет оператор ROLLBACK.

Полное определение понятия транзакции невозможно без указания основных свойств данного механизма, которые получили название ACID-свойств:

- атомарность (англ. atomicity) – выполняется все или ничего;
- согласованность (англ. consistency) – транзакции переводят БД из одного непротиворечивого состояния в другое, тем самым сохраняя БД в согласованном состоянии;
- изолированность (англ. isolation) – транзакции изолированы друг от друга;
- долговечность (англ. durability) – зафиксированные транзакции сохраняют свои изменения в БД постоянно, т.е. происходит

сохранение данных в физических файлах, расположенных в памяти ЭВМ и даже, если в следующий момент произойдет системный сбой, данные будут сохранены (за исключением случаев отказа аппаратного обеспечения, например, поломки головки дискового накопителя).

1.4.2. Три проблемы параллельности

При многопользовательском доступе к разделяемым ресурсам БД имеет место параллельный доступ к данным. Применительно к СУБД термин «параллельность» подразумевает возможность системы одновременной обработки многих транзакций, получающих доступ к одним и тем же данным в одно и то же время [**8**]. Проблемы параллельности будем рассматривать на основе примеров, в которых файлы данных состоят из элементов данных, каждый из которых представляет некоторый объект из предметной области без опоры на конкретную модель данных или физическую реализацию.

Известны три проблемы, возникающие при параллельной обработке данных:

- потеря результатов обновления – происходит обновление не тех данных, которые находятся в БД, а тех данных, которые были считаны до выполнения последнего обновления в БД. Проблема представлена ниже (табл. 3): обновления, выполненные транзакцией А теряются, так как извлечение данных транзакцией В было произведено, до этих обновлений;

Таблица 3

Потеря результатов обновления

Момент времени				
Транзакция	T1	T2	T3	T4
A	Извлечение элемента данных r		Обновление элемента данных r	
B		Извлечение элемента данных r		Обновление элемента данных r

- зависимость от незафиксированных результатов (табл.4) – происходит обновление тех данных, которые еще не были зафиксированы как постоянные в БД. Обновленные транзакцией А данные не

могут быть использованы любой другой транзакцией B до выполнения их фиксации (табл.4);

Таблица 4

Зависимость от незафиксированных результатов

Транзакция	Момент времени		
	T1	T2	T3
A	Обновление элемента данных r		Отмена транзакции
B		Извлечение элемента данных r	

- несогласованная обработка данных. Для рассмотрения проблемы несогласованной обработки данных обратимся к примеру простейшей биллинговой системы, элементы данных в которой содержат сведения об остатках на счетах абонентов. В процессе работы системы над данными выполняются две транзакции (табл.5): A – суммирует текущие остатки на счетах трех абонентов (ACC1 + + ACC2 + ACC3); B – выполняет перевод суммы в 10 рублей со счета ACC3 на счет ACC1. На счетах абонентов (ACC1, ACC2, ACC3) в начале выполнения транзакций находится 40, 50 и 30 рублей соответственно. Очевидно, что полученная в результате выполнения транзакции A сумма (110) является неверной. Отличие этого примера от предыдущего в том, что используемые транзакцией A данные в конечном итоге фиксируются (т.е. не происходит отката). Проблема возникает из-за того, что часть данных, участвующих в расчете суммы (транзакция A), изменяется уже после их учета (транзакция B).

Таблица 5

Несогласованная обработка данных

Транзакция	Момент времени							
	T1	T2	T3	T4	T5	T6	T7	T8
A	Извлечение элемента ACC1: Sum = 40	Извлечение элемента ACC2: Sum = 90						Извлечение элемента ACC3 Sum = 110 (а не 120)
B			Извлечение элемента ACC3	Обновление элемента ACC3: 30 → 20	Извлечение элемента ACC1	Обновление элемента ACC 1 40 → 50	Фиксация результатов	

Для решения указанных проблем параллельной обработки в СУБД применяются различные методологии. Ниже рассмотрены два популярных механизма управления конкурентным доступом к данным: с помощью блокировок и с применением так называемой многоверсионности.

1.4.3. Блокировки

Механизм блокировки представляет собой запрет на доступ к объекту базы данных (запись, кортеж, документ, узел и т.д.) в случае, если уже имеет место доступ к этому объекту из выполняющейся транзакции (т.е. транзакции, которая не была завершена к текущему моменту фиксацией или откатом).

Механизм блокировок в СУБД подразумевает два вида блокировок:

- *эксклюзивная блокировка* (X-блокировка) – не допускает совместного доступа;
- *разделяемая блокировка* (S-блокировка) – допускает совместный доступ на чтение.

Механизм блокировки определяется следующим образом [8]:

1. Если транзакция установила эксклюзивную блокировку на запись r, то запрос любой другой транзакции на блокировку этой же записи будет отменен.

2. Если транзакция установила разделяемую блокировку на запись r, то:

а) запрос со стороны любой другой транзакции на установление эксклюзивной блокировки на запись r будет отклонен;

б) запрос со стороны любой другой транзакции на установление разделяемой блокировки на запись r будет удовлетворен.

Таким образом, допускается параллельный доступ к данным в БД на чтение, однако, в случае необходимости выполнения обновления данных требуется, чтобы только одна транзакция имела доступ к обновляемым данным. В случае, если транзакция A уже установила S-блокировку записи r и при этом никакая другая транзакция еще не запросила S-блокировку записи r, то транзакция A может расширить блокировку до эксклюзивной.

Известной проблемой механизма блокировок является взаимная блокировка данных, которая происходит в случае, когда две или более транзакций попадают в режим ожидания снятия блокировки на

одни и те же данные. В этом случае СУБД должна устранить взаимную блокировку путем принудительной отмены одной или нескольких ожидающих транзакций.

1.4.4. Многоверсионность данных

Методология управления конкурентным доступом с помощью многоверсионности (анг. MVCC – MultiVersion Concurrency Control) впервые получила свое развитие в конце 70-х, начале 80-х годов XX века. В основе идеи MVCC лежит возможность существования в БД нескольких версий одного и того же элемента данных, что позволяет приложениям различных видов эффективно использовать СУБД:

- приложения, подавляющее число запросов в которых ориентировано на выборку данных, могут работать со своей версией изменяемых в настоящий момент данных, что исключает необходимость их эксклюзивной блокировки, а, следовательно, позволяет упростить логику функционирования системы и повысить скорость выполнения запросов;
- приложения, которые ориентированы на совместную работу с данными (модификацию и чтение), могут существенно улучшить временные показатели взаимодействия с СУБД для случаев, когда происходит любое количество параллельных чтений и не более одной записи разделяемых данных. В этом случае использование MVCC позволяет избежать возникновения взаимных блокировок, однако, в случае параллельного изменения данных данный метод не столь эффективен.

Таким образом, многоверсионность позволяет обеспечивать параллельный доступ к данным за счет предоставления каждому пользователю версии (снимка) БД. Основным достоинством метода по сравнению с рассмотренным выше механизмом блокировок является возможность добиться того, чтобы операции модификации данных не блокировали доступ к этим данным на чтение, и наоборот, чтобы читающие транзакции не блокировали пишущих.

Пример, изображающий концепцию метода, представлен ниже (рис. 6): транзакция t_1 в процессе своего выполнения читает элемент данных (например, запись) x из некоторого файла (таблицы) A. Затем транзакция t_2 изменяет x, но для того чтобы не блокировать транзакцию t_1 и не дожидаться ее завершения транзакция t_2 создает новую версию x_2. После того как транзакция t_2 создала «свою» вер-

сию элемента данных x_2, в случае, если потребуется чтение (или запись) x в транзакции t_2 будет прочитана (записана) версия x_2.

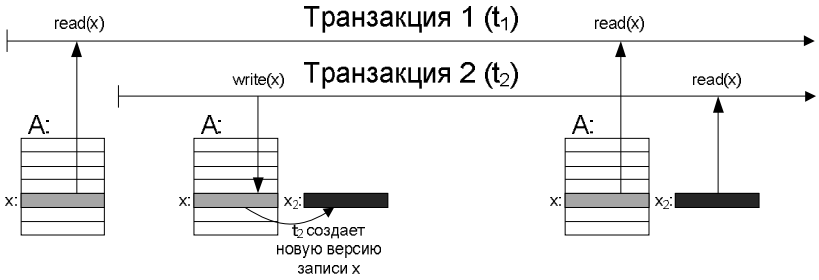


Рис.6. Суть метода многоверсионного управления

При реализации многоверсионности используется дельта-компрессия для оптимизации пространства под хранимые версии записей, при которой новая версия записи целиком не сохраняется. Вместо этого в качестве новой версии записи сохраняется перечень отличий от предыдущей версии.

Известен ряд алгоритмов обеспечения многоверсионности [21]:

- использование временных меток (англ. MVTO – MultiVersion Timestamp Ordering);
- многоверсионный двухфазный протокол синхронизации (англ. MV2PL – MultiVersion Two-Phase Locking Protocol) и его модификация – 2V2PL;
- многоверсионный протокол для читающих транзакций (англ. ROMV – MultiVersion Protocol for Read-Only Transactions);
- управление многоверсионностью на основе графа конфликтов (англ. MVSG – MultiVersion Serialization Graph Testing).

Наибольшей популярностью на практике пользуются алгоритмы 2V2PL и ROMV (например, ROMV реализуется в подсистеме InnoDB, используемой MySQL). Рассмотрим многоверсионный протокол для транзакций, не изменяющих данные (ROMV). Этот протокол является гибридным, т.е. объединяет в себе два подхода (MVTO и 2V2PL) и ориентирован, прежде всего, на приложения, для которых важна скорость выполнения читающих транзакций (т.е. транзакций не производящих изменение данных).

Перед рассмотрением алгоритма требуется определить ряд понятий. Будем считать, что один и тот же элемент данных может

существовать в БД в нескольких *версиях*. Различают следующие версии элемента данных:

- *завершенные версии* (committed versions) – версии, созданные транзакциями, которые уже успешно закончили свою работу;
- *текущие версии* (current versions) – последние (в хронологическом порядке) из завершенных версий;
- *незавершенные версии* (uncommitted versions) – версии, созданные теми транзакциями, которые все еще находятся в работе.

Для описания алгоритма введем ряд обозначений:

- t_i – транзакция;
- $ts(t_i)$ – временная метка (timestamp), полученная транзакцией в начале своей работы;
- $r_i(x)$ – операция чтения транзакцией t_i элемента данных x;
- $r_i(x_k)$ – операция чтения транзакцией t_i версии элемента данных x, созданную транзакцией t_k;
- $w_i(x)$ – операция записи транзакцией t_i версию элемента данных x.

В процессе работы алгоритм ROMV использует три вида блокировок:

- блокировка на чтение (rl – read lock) устанавливается на текущую версию элемента x непосредственно перед прочтением;
- блокировка на запись (wl – write lock) устанавливается перед созданием новой (незавершенной) версии x;
- блокировка на фиксацию (cl – commit lock) устанавливается перед выполнением последней операции транзакции по отношению к каждому элементу данных, который был записан.

Ниже представлена табл.6 совместимости блокировок, на которой знаком «+» обозначены блокировки, которые могут быть параллельно установлены на один элемент данных, а знаком «-» – несовместимые блокировки.

Таблица 6

Совместимость блокировок

	RL	WL	CL
RL	+	+	–
WL	+	–	–
CL	–	–	–

Во время своей работы ROMV-планировщик разделяет все транзакции на две группы: *запросы* (queries) и *транзакции, изме-*

няющие данные (update transactions), при этом транзакции обрабатываются следующим образом:

1. Транзакции, изменяющие данные, обрабатываются в соответствии с протоколом S2PL, который является модификацией 2V2PL – все монопольные блокировки снимаются лишь в конце транзакции:

1.1. если операция не является *финальной* (последней операцией транзакции перед ее завершением), то операция w(x) выполняется только после завершения транзакции, записавшей последнюю версию x;

1.2. если операция является финальной для транзакции t_i, то она откладывается до тех пор, пока не завершатся следующие транзакции:

- все транзакции t_j, прочитавшие текущую версию данных, которую должна заменить версия, записанная t_i (устраняется возможность неповторяющегося чтения, т.е. чтения при котором в рамках одной транзакции будут прочитаны две различные версии одних данных);

- все транзакции t_j, которые записали версии, прочитанные t_i;

2. Запросы обрабатываются так же, как и в протоколе MVTO:

2.1. планировщик преобразует операцию r_i в операцию $r_i(x_k)$, где x_k – это версия элемента x, помеченная наибольшей временной меткой $ts(t_k)$, для которой справедливо: $ts(t_k) \leq ts(t_i)$, при этом для каждого запроса временная метка устанавливается временем начала транзакции, что отличает данный подход от протокола MVTO;

2.2. завершение транзакции t_i откладываются до того момента, когда завершатся все транзакции, записавшие версии данных, прочитанных t_i.

Несмотря на то, что идеи алгоритма ROMV достаточно просты, при реализации данного подхода требуется наличие специального компонента в СУБД – «сборщика мусора», который удаляет ненужные версии данных.

1.4.5. Распределенная обработка данных

С точки зрения основной цели применения БД, а именно – разработки и функционирования прикладных программ и автоматизированных систем – различают следующие виды архитектуры систем БД, предназначенных для многопользовательского доступа:

- *файл-серверная архитектура*, при которой хранение данных и кода программы производится в файле (или файлах) БД на сервере, работа с ним осуществляется процессором рабочей станции с помощью сетевых коммуникаций, как правило, в рамках предоставляемого операционной системой сервера API, но вся обработка данных происходит исключительно на стороне клиента;
- *клиент-серверная архитектура*, при которой операции над данными распределены между источником данных (сервером) и пользовательскими приложениями (клиентами), взаимодействующими с сервером через компьютерную сеть.

Файл-серверная архитектура имеет существенные ограничения по числу одновременных подключений к серверу БД (не рекомендуется более 30 пользователей, на практике редко встречаются системы с числом пользователей более 10), кроме того, рост числа пользователей существенным образом сказывается на падении производительности системы (возрастает сетевая нагрузка, уменьшается скорость обработки данных в БД и т.д.). С точки зрения проектирования программных средств и АИС ориентированных на большое число пользователей и/или относительно высокую нагрузку (пусть даже в перспективе) более предпочтительным является использование СУБД, работающих по клиент-серверной архитектуре.

К основным преимуществам использования клиент-серверной архитектуры относят: снижение требований к клиентским компьютерам, так как большая часть вычислений выполняется на сервере, повышенный уровень безопасности за счет хранения всех данных в одном месте, распределение вычислительной нагрузки между различными ЭВМ. С другой стороны, в случае отказа сервера перестает работать вся система, что является недостатком.

С точки зрения БД различают двух- и трехзвенную архитектуру клиент-серверных приложений. В двухуровневой архитектуре сервером является сама СУБД, а клиентское приложение взаимодействует с СУБД, используя специальные драйвера (программные модули). Варианты распределения основных функций между клиентским приложением и сервером БД, представляемых СУБД, в двухзвенной модели проиллюстрированы ниже (рис.7).

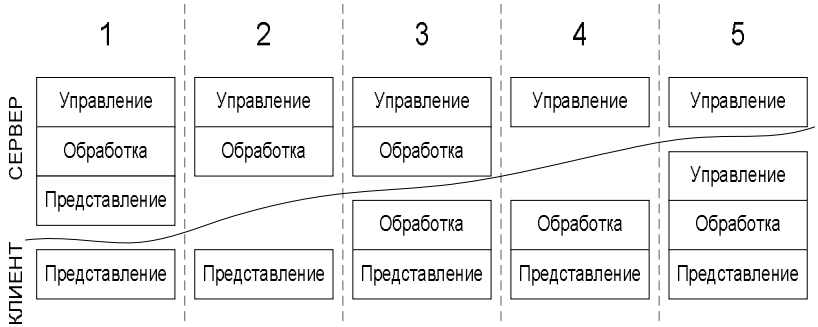


Рис.7. Двухзвенная модель распределения функций
в архитектуре клиент-сервер

На рис.7 представлены пять вариантов разделения основных функций по взаимодействию с СУБД: управление данными, обработка данных и представление данных между клиентской и серверной частями приложения:

1) *распределенное представление*: клиентское приложение практически полностью вырождено (терминальное приложение) и требуется только для отображения экранов (сформированных на сервере) конечному пользователю;

2) *удаленное представление*: клиентское приложение отвечает за представление данных, обработка и управление которыми происходит на стороне СУБД (сервер БД);

3) *распределенная функция*: часть функций по обработке полученных из БД по средствам СУБД данных выполняется на клиенте;

4) *удаленный доступ*: СУБД отвечает только за управление данными, а вся бизнес-логика функционирования АИС, обработки и представления данных реализована в клиентских приложениях;

5) *распределенное управление* (*система*): функции сервера БД вырождены (либо практически не используются), при этом каждый клиент способен самостоятельно управлять частью данных в распределенной БД без обращения к серверу.

В трехзвенной модели взаимодействия между СУБД и клиентским приложением функционирует приложение-сервер. Таким образом, каждая из трех функций приложения реализуется на отдельном компьютере (рис.8).

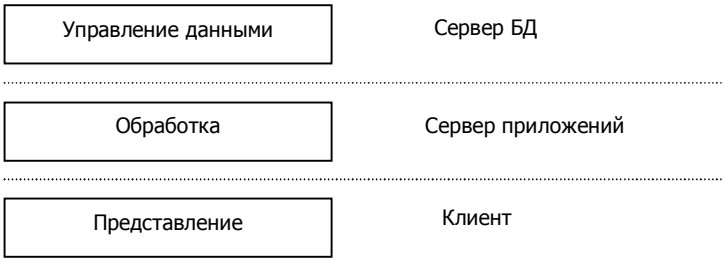


Рис.8. Трехзвенная модель распределения функций
в архитектуре клиент-сервер

Данная модель имеет название – модель сервера приложений (англ. AS – Application Server) [13], так как именно сервер приложений играет центральную роль в данной модели, реализуя основную часть бизнес-логики системы.

Так как преимущества и недостатки двух- и трехуровневой архитектуры ассиметричны, здесь приведем преимущества и недостатки только трехуровневой архитектуры.

Преимущества:

- масштабируемость – потенциальная возможность АИС к относительно простому расширению за счет наращивания аппаратных и программных ресурсов, призванная удовлетворить рост рабочей нагрузки;
- конфигурируемость: так как физически уровни в трехуровневой модели изолированы, то становится возможным относительно простое переконфигурирование отдельных частей ИС;
- высокий потенциально достижимый уровень безопасности:

- отсутствует прямой (открытый) доступ к серверу БД со стороны клиентских приложений, таким образом, проще контролировать безопасность в СУБД;

- существует возможность реализации дополнительных средств защиты канала между сервером и клиентами специальными методами и средствами (АЦП, шифрование, многофакторная аутентификация и т.д.);

- повышенная надежность системы: функционирование отдельных клиентских приложений не зависит друг от друга, с другой

стороны, могут одновременно функционировать несколько взаимозаменяемых серверов приложений;

- низкие требования к скорости канала между клиентами и сервером с учетом того, что основная часть обработки выполняется на сервере;
- требования к производительности и техническим характеристикам клиентских ЭВМ невысоки, так как в их функции входит только представление данных конечным пользователям.

Недостатки:

- относительно более высокая сложность создания приложений, обусловленная необходимостью проектирования двух уровней приложения (сервер и клиент) и протокола взаимодействия между ними;
- каждый из уровней требует специальных действий по настройке приложения, а также по его сопровождению, вследствие чего разворачивание и администрирование усложняется;
- так как основные функции по управлению и обработке данных выполняют сервер приложений и сервер БД, то при большом числе пользователей требования к производительности этих узлов существенно возрастают (для решения этой проблемы применяют распределенные приложения);
- высокие требования к пропускной способности канала между сервером приложений и сервером БД.

Типичным примером реализации трехзвенной архитектуры является динамическое web-приложение, в котором с использованием серверных скриптов (например, PHP) организовано взаимодействие с СУБД (например, MySQL). При этом в качестве сервера приложений выступает web-сервер (например, Apache), а в качестве тонкого клиента любой web-браузер.

1.5. Вопросы для самоконтроля

1. База данных: определения в узком и широком смысле, преимущества перед «бумажными» методами хранения информации, варианты классификации.
2. Трехуровневая модель ANSI/SPARC: цель использования, описание уровней.
3. Система управления базами данных: определение и назначение.

4. Каково качественное отличие СУБД от предшествующих систем автоматизированной обработки информации, в чем его суть и преимущества?

5. Что понимается под целостностью базы данных? Что такое ограничение целостности, как они используются в СУБД и для чего?

6. Каковы основные функции СУБД?

7. Какие основные компоненты в СУБД принято выделять и каким образом они представлены в системе?

8. Администратор и его функции в отношении базы данных.

9. Какие основные модели данных существуют, в чем их достоинства и недостатки?

10. Что такое инфологическое проектирование и каким образом оно выполняется?

11. Каким образом реализуется даталогический и физический проект базы данных?

12. Что такое транзакция в приложении в теории БД и какие важнейшие свойства транзакций существуют?

13. Приведите примеры наиболее известных проблем параллельной обработки данных.

14. Блокировка в БД: виды блокировок, механизм и основная проблема.

15. Каковы суть метода управления разделяемыми данными на основе многоверсионности и его преимущества по отношению к методу использования блокировок? Какие протоколы управления версиями в БД используются?

16. Какие виды архитектуры систем БД существуют? В чем их достоинства и недостатки?

17. Опишите двухзвенную и трехзвенную модели распределения функций в архитектуре клиент-сервер.

18. Преимущества и недостатки трехуровневой архитектуры.

2. РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ: ПРОЕКТИРОВАНИЕ, СОЗДАНИЕ, ИСПОЛЬЗОВАНИЕ

2.1. Реляционная модель данных

2.1.1. Основные определения

Информационная модель предметной области представляет собой множество объектов реального мира. *Объект* описывает нечто существенное (материальное или нематериальное) с точки зрения хранения данных и характеризуется определенным набором *свойств*, которые также называют *атрибутами*. Очевидно, что объекты с одинаковым набором атрибутов, для которых значения этих атрибутов совпадают, не могут быть формально идентифицированы (различены) никакой АИС, т.е. с точки зрения БД такие объекты несут одинаковую семантику и, следовательно, идентичны. Таким образом, каждый описываемый объект должен быть существенным и различимым от других объектов.

Атрибут имеет *имя* (некоторый символьный идентификатор) и может иметь *значения*. *Тип объекта* определен совокупностью имен атрибутов, тогда как каждый объект (экземпляр) характеризуется совокупностью значений своих атрибутов. При этом атрибут принимает значения из предопределенного допустимого множества, называемого *доменом* этого атрибута.

Упорядоченная последовательность значений атрибутов, характеризующих некоторый объект из предметной области, представляет *кортеж*. Таким образом, кортеж в реляционной модели представляет некоторый объект реального мира, из чего следует, что каждый кортеж также существенен и различим.

Упорядоченная последовательность пар (имя атрибута; имя домена), определяющая некоторый тип объекта из предметной области, представляет *схему отношения*. *Схема БД* в реляционной модели состоит из совокупности всех представляемых БД схем отношений. Тогда физический проект БД в упрощенном виде может быть представлен как множество отношений (в действительности, требуется еще как минимум определение ограничений целостности).

Отношение – это базовое понятие реляционной модели, которое определяется как «конечное множество кортежей, составленных из допустимых значений атрибутов схемы отношений» [13]. Тео-

ретико-множественное определение отношения звучит так: подмножество прямого произведения доменов (одного и более) атрибутов. *Арность отношения* – число атрибутов, входящих в схему отношения.

Различают *переменную отношения* и *значение переменной отношения*. Переменная отношения определяется именем (символьный идентификатор) и схемой отношения. Значение переменной отношения (значение отношения) представляется схемой отношения и множеством кортежей, принадлежащих отношению в некоторый момент времени и называемых *телом отношения*.

С практической точки зрения удобно представлять отношение в виде таблицы (рис.9).

Таблица (отношение) - Люди

Названия атрибутов
Типы данных
Домены для каждого атрибута

Фамилия	Имя	Пол	Дата рождения
Строка 50	**Строка 20**	**Символ**	**Дата**
Строка символов русского алфавита не более 50 символов	Строка символов русского алфавита не более 50 символов	‘М’ или ‘Ж’	01.01.1900 до текущей
Алданов	Григорий	М	12.10.1990
Байсанханов	Мурат	М	01.04.1991
Величко	Виктория	Ж	08.08.1990
Гладышук	Елена	Ж	14.08.1990
Пугачева	Людмила	Ж	21.03.1989
Румянцева	Екатерина	Ж	05.09.1991
Хашиев	Илес	М	25.12.1990

Данные об объектах БД: Выделен кортеж (хранимая запись), который соответствует одному объекту из отношения «Люди». Он состоит из хранимых полей

этот столбец таблицы – совокупность значений атрибута *Пол*

Рис.9. Табличное представление понятий реляционной модели

Таким образом, можно установить следующие соответствия между реляционным отношением и его физической реализацией в виде таблицы:

- каждая строка (иначе, *хранимая запись*) таблицы представляет некоторый объект и соответствует понятию «кортеж»;
- строка состоит из *хранимых полей*, каждое из которых представляет значение соответствующего атрибута;
- название колонки таблицы соответствует имени атрибута;
- совокупность кортежей (хранимых записей) представляет тело отношения;

- типы данных для колонок таблицы определяются в соответствии с доменами атрибутов;
- совокупность названий колонок таблицы и соответствующих типов данных представляет схему отношения.

Для идентификации кортежа в отношении применяется понятие «ключ». *Ключ* – это непустое подмножество множества имен атрибутов из схемы отношения, значения которых уникальным образом идентифицируют кортеж при любом допустимом значении переменной отношения. Иными словами, для любого допустимого множества кортежей отношения (значение переменной отношения), любые два различных кортежа обязательно отличаются по значениям хотя бы одного атрибута, входящего в ключ.

Над отношениями можно выполнять *реляционные операции*, при этом значением любой реляционной операции также является отношение. С помощью реляционных операций могут быть получены *реляционные выражения*.

Следует отметить два важных свойства реляционной модели данных, обуславливающих простоту ее понимания и физической реализации:

- модель является логической, то есть отношения являются логическими (абстрактными), а не физическими (хранимыми) структурами;
- всё информационное наполнение базы данных представлено одним и только одним способом, а именно — явным заданием значений атрибутов в кортежах отношений, т.е. не существует указателей и подобных им типов данных, связывающих одно значение с другим.

2.1.2. Виды реляционных отношений

Выделяют следующие виды отношений, используемые в реляционных системах баз данных [**8**]:

- *именованное отношение* – отношение, с которым ассоциировано некоторое имя;
- *неименованное отношение* – отношение, для которого задана только его схема и (возможно) тело, но имя не определено;
- *производное отношение* – это отношение, которое определено через другие именованные отношения, являющиеся для него базовыми посредством реляционного выражения;

- *базовое отношение* – это именованное отношение, которое не является производным от других отношений;
- *выражаемое отношение* – это отношение, которое можно получить из некоторого набора отношений по средствам реляционного выражения;
- *обновляемое отношение* – это производное отношение, чье тело изменяется в соответствии с изменениями базовых отношений;
- *представление* – это именованное обновляемое отношение;
- *снимок* – это именованное производное отношение, которое не является обновляемым, т.е. значение переменной отношения фиксируется на некоторый момент времени, а его изменение не связано с изменениями в базовых отношениях (снимки могут совсем не обновляться, либо изменяться периодически, например, раз в день);
- *хранимое отношение* – это отношение, которое поддерживается во внешней памяти (все базовые отношения, а также снимки являются хранимыми отношениями);
- *результат реляционного запроса* – неименованное производное отношение, которое является результатом вычисления некоторого реляционного выражения (не является хранимым).

Реализация представлений на физическом уровне часто сводится к ассоциированию с именем представления некоторого реляционного выражения (реляционного запроса), которое вычисляется при выполнении доступа к представлению. С целью увеличения производительности СБД может выполняться кэширование результатов вычисления реляционного выражения [10], которые сохраняются во внешней памяти в виде снимка. При этом, как только происходит модификация данных хотя бы в одном из базовых отношений представления, снимок должен быть обновлен. Таким образом, с одной стороны, поддерживается актуальность данных в представлении, с другой, – повышается производительность СБД за счет использования методологии кэширования. Такие представления часто называют *материализованными представлениями* (англ. materialized view).

2.1.3. Реляционная алгебра

Рассмотрим операции, определенные над отношениями в рамках реляционной модели данных. Так как отношение является

множеством, то для него применимы все операции, известные для множеств, а также некоторые специфичные операции:

- *операция объединения*: пусть R и S – отношения арности n, тогда их объединение – это отношение U арности n, составленное из кортежей $k=(v_1, v_2, \dots v_n)$, таких что v_i – значение i-го атрибута кортежа k и при этом каждый из кортежей k входит, по меньшей мере, в одно из базовых отношений (R или S):

$$U = R \cup S = \{ (v_1, v_2 \dots v_n) \mid (v_1, v_2 \dots v_n) \in R \Subset (v_1, v_2 \dots v_n) \in S \};$$

- *операция разности отношений*: пусть R и S – отношения арности n, тогда разность отношений (R-S) – это отношение D арности n, составленное из кортежей $k=(v_1, v_2, \dots v_n)$, таких что v_i – значение i-го атрибута кортежа k и при этом каждый из кортежей k входит в отношение R и одновременно не входит в отношение S:

$$D = R - S = \{ (v_1, v_2 \dots v_n) \mid (v_1, v_2 \dots v_n) \in R \in (v_1, v_2 \dots v_n) \notin S\};$$

- *операция декартового произведения* (прямое произведение двух отношений): пусть R – отношение арности n, а S – отношение арности m, тогда декартовое произведение отношений (R×S) – это отношение P арности n+m, составленное из всех кортежей $k=(v_1, v_2, \dots v_n, w_1, w_2 \dots w_n)$, таких что v_i – значение i-го атрибута кортежа r из отношения R, w_j – значение j-го атрибута кортежа s из отношения S:

$$P = R \times S = \{(v_1 \dots v_n, w_1 \dots w_m) \mid (v_1 \dots v_n) \in R \in (w_1 \dots w_m) \in S\};$$

- *операция проекции*: пусть R – отношение арности n, тогда проекция отношения R – это отношение $\pi_{i1,i2\dots im}(R)$, полученное из R путем проецирования его на схему отношения $(i_1, i_2, \dots i_m)$, где i_j – имя атрибута из схемы отношения R, при этом справедливо:

$$\pi_{i1,i2\dots im}(R) = \{(a_1 \dots a_m) \mid (b_1 \dots b_n) \in R \mid a_j = b_{ij} \;\; j = 1 \dots m \};$$

- *операция селекции*: пусть R – отношение арности n, F – предикат на множестве атрибутов отношения R, т.е. F возвращает одно из двух значений: истину (1) или ложь (0), тогда результат выполнения селекции – это отношение $\delta_F(R)$, которое включает только те кортежи из отношения R, для которых предикат F принимает значение «истина»:

$$\delta_F(R)=\{(v_1, v_2 \dots v_n) \in R \mid F \equiv 1\};$$

- *естественное соединение*: пусть R – это отношение с атрибутами $(X_1, X_2, \dots X_n, Y_1, Y_2, \dots Y_m)$, а S – отношение с атрибутами $(Y_1, Y_2, \dots Y_m, Z_1, Z_2, \dots Z_k)$, т.е. атрибуты $Y_1, Y_2, \dots Y_m$ и только они являются общими для этих отношений, тогда естественное соединение отношений (R Join S) – это отношение с атрибутами $(X_1, X_2, \dots X_n, Y_1,$

Y_2, … Y_m, Z_1, Z_2, … Z_k), составленное из кортежей (x_1, x_2, … x_n, y_1, y_2, … y_m, z_1, z_2, … z_k) из R и S, таких, что в отношении R: $X_1=x_1$, $X_2=x_2$, … $X_n=x_n$, $Y_1=y_1$, $Y_2=y_2$, … $Y_m=y_m$ и при этом в отношении S: $Y_1=y_1$, $Y_2=y_2$, … $Y_m=y_m$, $Z_1=z_1$, $Z_2=z_2$, … $Z_n=z_n$.

Ниже рассматриваются примеры применения реляционных операций для заданных значений переменных отношений R и S:

R:

A	**B**	**C**
A	B	C
D	A	F
C	B	D

S:

D	**E**	**F**
B	G	A
D	A	F

Объединение R и S ($R \cup S$):

A	**B**	**C**
A	B	C
D	A	F
C	B	D
B	G	F

Декартово произведение ($R \times S$):

A	**B**	**C**	**D**	**E**	**F**
A	B	C	B	G	A
D	A	F	B	G	A
C	B	D	B	G	A
A	B	C	D	A	F
D	A	F	D	A	F
C	B	D	D	A	F

Разность R и S (R–S):

A	**B**	**C**
A	B	C
C	B	D

Проекция $\pi_{A,C}$ (R):

A	**C**
A	C
D	F
C	D

Селекция $\delta_{B=B}$(R):

A	**B**	**C**
A	B	C
C	B	D

Анализируя пример операции «естественное соединение», введем в рассмотрение отношения A = {код, имя, статус, город} – поставщики и B = {номер, вес, город} – детали. Значения переменных отношений A и B:

Отношение A:

Код	**Имя**	**Статус**	**Город**
1	Иванов	20	Москва
2	Петров	10	Казань
3	Сидоров	30	Казань
4	Семенов	20	Москва
5	Конкин	30	Новгород

Отношение B:

Номер	Вес	Город
1	12	Москва
2	17	Казань
3	17	Ростов
4	14	Москва
5	12	Казань
6	19	Москва

Отношение A Join B:

Код	Имя	Статус	Город	Номер	Вес
1	Иванов	20	Москва	1	12
1	Иванов	20	Москва	4	14
1	Иванов	20	Москва	6	19
2	Петров	10	Казань	2	17
2	Петров	10	Казань	5	12
3	Сидоров	30	Казань	2	17
3	Сидоров	30	Казань	5	12
4	Семёнов	20	Москва	1	12
4	Семёнов	20	Москва	4	14
4	Семёнов	20	Москва	6	19

Ранее мы условились, что БД может быть представлена в качестве множества отношений, после чего на этом множестве отношений были определены операции такие, что каждый операнд и результат этих операций является отношением. Таким образом, нами определена *реляционная алгебра*, при этом данная система является замкнутой, так как никакая из операций не приводит к выходу из множества отношений. На базе представленных операций реляционной алгебры строятся основные команды языка SQL, которые будут рассмотрены в разделе 2.3 со ссылкой на соответствующие операции из этого параграфа.

2.1.4. Правила Кодда

В своих работах Э. Кодд сформулировал ряд требований к реляционным СУБД, которые были названы в его честь правилами Кодда [2, 3]. В действительности предложенные правила полностью

не соблюдаются практически ни одной из существующих, так называемых, реляционных СУБД.

Кроме того, что все реляционные СУБД должны представлять данные на физическом уровне в качестве двумерных таблиц, перед реляционной системой выдвигаются следующие требования [**3**]:

Основное правило (Foundation rule): реляционная СУБД должна быть способна полностью управлять БД, т.е. использовать исключительно свои реляционные возможности для управления БД.

1. **Явное представление данных** (Information rule): информация должна быть представлена в виде атомарных данных, хранящихся в ячейках таблицы, при этом порядок строк в таблице не должен влиять на смысл данных.

2. **Гарантированный доступ к данным** (Guaranteed Access Rule): к каждому элементы данных должен быть гарантирован доступ с помощью комбинации имени таблицы, первичного ключа строки и имени столбца.

3. **Полная обработка неопределенных значений** (Systematic Treatment of Null Values): неизвестные значения (null-значения) должны поддерживаться для всех типов данных при выполнении любых операций.

4. **Доступ к базе данных в терминах реляционной модели** (Active On-Line Catalog Based on the Relational Model): словарь данных должен храниться в форме реляционных таблиц и СУБД должна поддерживать доступ к нему с использованием стандартных языковых средств.

5. **Полнота подмножества реляционного языка** (Comprehensive Data Sublanguage Rule): СУБД должна поддерживать хотя бы один реляционный язык, который имеет линейный синтаксис, может использоваться как интерактивно (пользователем СУБД), так и в прикладных программах, а также поддерживает все операции, необходимые для определения данных, определения представлений, манипулирование данными, управления ограничениями целостности, управление доступом и транзакциями.

6. **Обновляемость представлений** (View Updating Rule): каждое представление должно поддерживать все операции манипулирования данными, которые поддерживаются реляционными таблицами (выборка, вставка, модификация и удаление).

7. **Наличие высокоуровневого языка** (High-Level Insert, Update, and Delete): операции вставки, обновления и удаления должны поддерживаться к множеству строк любой мощности.

8. **Физическая независимость данных** (Physical Data Independence): приложения не должны зависеть от используемых способов хранения данных и аппаратного обеспечения.

9. **Логическая независимость данных** (Logical Data Independence): представление данных в приложениях не должно зависеть от структуры реляционных таблиц (необходимо использовать представления).

10. **Независимость контроля целостности** (Integrity Independence): информация, необходимая для поддержания БД в целостном состоянии, должна находиться в словаре данных, при этом целостность данных должна поддерживаться автоматически.

11. **Дистрибутивная независимость** (Distribution Independence): БД может быть распределенной, находиться на нескольких компьютерах, переноситься с одного компьютера на другой, но все это не должно оказывать влияния на работоспособность приложений.

12. **Согласование языковых уровней** (The Nonsubversion Rule): использование языка низкого уровня не должно приводить к игнорированию правил безопасности и ограничений целостности, которые поддерживаются языком более высокого уровня.

2.2. Формальный подход к проектированию реляционных баз данных

В основе проектирования реляционных баз данных лежат идеи нормализации, которые предоставляют научно строгие и обоснованные критерии качества проекта БД, а также формальные методы для усовершенствования этого качества. При этом практически во всей сфере информационных технологий отсутствуют формальные методы оценки и улучшения качества проектных решений, сопоставимые с теорией нормализации по уровню формальной строгости [12].

В данном разделе приведены основные аспекты, позволяющие формально ответить на вопрос о рациональности выбора той или иной схемы реляционной БД, включающей схемы отношений и поддерживаемые ограничения целостности. Соблюдение этих прин-

ципов гарантирует возможность масштабирования, а также позволяет избежать известных коллизий, о которых будет сказано ниже. В реальных проектах иногда бывает выгодно с точки зрения производительности системы БД и/или прикладных программных средств жертвовать некоторыми из представленных принципов, однако, такие жертвы требуют глубоких знаний о функционировании конкретной СУБД, архитектуре взаимодействующих с ней приложений, а также о возможных ошибках в структуре БД, которые нормализацией не устраняются.

2.2.1. Функциональные зависимости

Функциональная зависимость (ФЗ) – это тип зависимости, возникающий между атрибутами отношения. Различают функциональные зависимости первого и второго типов.

Функциональная зависимость первого типа

Пусть X и Y – произвольные (непустые) подмножества множества атрибутов отношения R. Говорят, что Y функционально зависит от X тогда и только тогда, когда в текущем значении переменной отношения R каждому значению X соответствует только одно значение Y.

Функциональная зависимость второго типа

Пусть X и Y – произвольные (непустые) подмножества множества атрибутов отношения R. Говорят, что Y функционально зависит от X тогда и только тогда, когда в каждый момент времени при любом значении переменной отношения R каждому значению X соответствует только одно значение Y.

Очевидно, что определение ФЗ второго типа представляется более ценным с точки зрения проектирования схемы БД, так как информация в БД меняется в процессе работы с ней, а значит, не имеет смысла создание ограничений целостности (в виде ФЗ), которые существуют только для некоторых наборов записей в реляционной таблице. Очевидно также, что все ФЗ 2-го типа удовлетворяют требованиям в ФЗ первого типа, но при этом не все ФЗ первого типа являются ФЗ второго типа.

ФЗ записываются в следующем виде: $\{X\}\rightarrow\{Y\}$, где X – это детерминант, а Y – зависимая часть. Фактически эта запись *означает* (**суть ФЗ**): если в некотором отношении R имеется два кортежа, ко-

торые совпадают по значениям атрибутов, составляющих множество X, то они **всегда** совпадают по значениям атрибутов множества Y.

Определение ФЗ неразрывно связано с семантикой предметной области. Например, рассмотрим отношение Поставки = {Код Поставщика, Город, Товар, Кол-во}. Если в отношении «Поставки» имеет место ФЗ {Код Поставщика}→{Город}, формально это означает, что в таблице не может появиться информация о поставках товара от одного и того же поставщика из разных городов. С точки зрения реалий предметной области, эта ФЗ означает, что один поставщик не может иметь филиалов (т.е. каждый поставщик располагается строго в одном городе).

Среди множества ФЗ выделяют *тривиальные* ФЗ, в которых зависимая часть ФЗ является подмножеством ее детерминанта.

2.2.2. Правила Армстронга и неприводимое множество ФЗ

В 1974 году Вильям Армстронг предложил набор правил вывода новых функциональных зависимостей на основе имеющихся. Пусть имеется отношение R и множества атрибутов A, B, C, D $\subseteq$ R. Для сокращения записи операцию объединения двух множеств X и Y будем записывать: XY. Армстронг определили следующие правила вывода:

- *рефлексивность*: (B $\subseteq$ A) $\Rightarrow$ (A $\rightarrow$ B) – если B – подмножество множество A, то B функционально зависит от A;
- *дополнение*: (A $\rightarrow$ B) $\Rightarrow$ (AC $\rightarrow$ BC) – если B функционально зависит от A, то дополнение (объединение) множества B на элементы множества C функционально зависит от дополнения множества A на элементы множества C;
- *транзитивность*: (A $\rightarrow$ B) $\wedge$ (B $\rightarrow$ C) $\Rightarrow$ (A $\rightarrow$ C);
- *самоопределение*: (A $\rightarrow$ A);
- *декомпозиция*: (A $\rightarrow$ BC) $\Rightarrow$ (A $\rightarrow$ B) $\wedge$ (A $\rightarrow$ C) – если множество BC функционально зависит от A, то полученные в результате декомпозиции подмножества B и C также функционально зависят от множества A;
- *объединение*: (A $\rightarrow$ B) $\wedge$ (A $\rightarrow$ C) $\Rightarrow$ (A $\rightarrow$ BC) – операция обратная декомпозиции;
- *композиция*: (A $\rightarrow$ B) $\wedge$ (C $\rightarrow$ D) $\Rightarrow$ (AC $\rightarrow$ BD) – если B функционально зависит от A и D функционально зависит от C, то

объединение множеств BD функционально зависит от объединения множеств AC;

- *теорема всеобщего объединения Дарвена*: $(A \rightarrow B) \wedge (C \rightarrow D) \Rightarrow (A \cup (C - B) \rightarrow BD)$.

Правила Армстронга применяются для преобразования одних функциональных зависимостей к другим. Пусть S – множество ФЗ отношения, из которых с использованием правил Армстронга могут быть получены другие ФЗ. Множество всех ФЗ, которые могут быть получены из множества S, называют *замыканием множества функциональных зависимостей* (S^+). Задача построения замыкания множества ФЗ является NP-полной (т.е. получение множества S^+ с помощью правил Армстронга является практически невозможным).

Так как ФЗ – это особый вид ограничений целостности, накладываемых на БД, то с практической точки зрения важно получить такое множество S_2, которое было бы эквивалентным заданному множеству S и при этом имело бы простую структуру для быстрой проверки ограничений при модификации данных в БД. В качестве такого множества в теории РБД принято выбирать неприводимое множество ФЗ. Заметим, что *эквивалентными* называются такие множества ФЗ, замыкания которых равны, т.е. $S \equiv S_2 \Leftrightarrow S^+ \equiv S_2^+$.

Неприводимое множество ФЗ может быть получено из исходного множества ФЗ путем последовательного его преобразования с использованием правил Армстронга к виду, в котором:

- зависимая часть каждой ФЗ простая, т.е. состоит из одного атрибута;
- в детерминанте каждой ФЗ не может быть опущен ни один атрибут без изменения замыкания множества ФЗ (если это условие выполняется, то говорят, что *ФЗ неприводима слева*);
- ни одна ФЗ не может быть опущена без изменения замыкания данного множества ФЗ.

2.2.3. Первичный ключ

Перед определением «первичный ключ отношения» рассмотрим другие виды ключей: потенциальные ключи и суперключи.

Потенциальный ключ (англ. CK – candidate key) отношения R – это непустое подмножество множества атрибутов отношения R, обладающее свойствами:

- *уникальности*, т.е. данное подмножество уникальным образом определяет кортеж в отношении R – каждому значению ключа может соответствовать только один кортеж в отношении R;
- *неизбыточности*, т.е. данное подмножество не содержит никакое другое подмножество, которое обладает свойством уникальности.

Суперключ отношения R – это непустое подмножество множества атрибутов отношения R, которое содержит как подмножество хотя бы один потенциальный ключ. Таким образом, суперключ обладает свойством уникальности, но является избыточным по отношению к своему подмножеству, являющемуся потенциальным ключом.

Для определения, является ли подмножество K множества атрибутов отношения R суперключом, необходимо проверить, обладает ли данное подмножество K свойством уникальности, т.е. являются ли остальные атрибуты отношения R функционально зависимыми от данного подмножества. Множество всех атрибутов из отношения R, функционально зависимых от подмножества атрибутов K, называется *замыканием множества атрибутов* K и обозначается K^+.

Рассмотрим суть метода на следующем примере. Пусть дано отношение R с множеством атрибутов A, множество функциональных зависимостей S отношения R и подмножество K множества атрибутов A. Чтобы проверить, что K обладает свойством уникальности (т.е. K является суперключом) построим замыкание множества атрибутов K. Обозначим замыкание множества атрибутов K через K^+. Напомним, что K^+ – это множество всех атрибутов из A, которые функционально зависят от K, а значит всегда выполняется $K \rightarrow K^+$. Рассмотрим случай, когда K^+=A, тогда $K \rightarrow A$. По определению ФЗ, если два кортежа совпадают по значениям атрибутов, имена которых содержатся в K, то они совпадают по значениям атрибутов, имена которых содержатся в A. Но A – это все атрибуты отношения R. Следовательно, получаем, если два кортежа из R совпадают по значениям из K, то они совпадают по значениям всех атрибутов, т.е. эти два кортежа **полностью совпадают**. Из определения реляционного отношения следует, что два одинаковых кортежа никогда не могут содержаться в отношении одновременно. Следовательно, если $K \rightarrow A$, тогда в любом значении отношения R никакие два кортежа не могут совпадать по значению атрибутов из K, т.е. K является суперключом и обладает свойством уникальности. Таким образом, **если полученное замыкание эквивалентно множеству всех атрибутов отношения R, то можем утверждать, что K обладает свойством уникальности**.

Алгоритм построения замыкания множества атрибутов K

Пусть имеется отношение R с множеством атрибутов A и множеством ФЗ $S=\{X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \ldots X_m \rightarrow Y_m\}$. Кроме того, дано множество атрибутов K ($K \subseteq A$). Построим множество всех атрибутов из A, которые функционально зависимы от K, т.е. построим замыкание множества атрибутов K^+. В качестве начального значения замыкания K^+ можем выбрать подмножество K, так как по правилу самоопределения $K \rightarrow K$, а значит справедливо: $K \subset K^+$:

$K^+ = K$
Цикл_пока истина
    j = 0, f = ложь
    Цикл_пока j < m
        j + +
        Если $(X_j \subset K^+) \wedge (y \mid y \in Y_j \wedge y \notin K^+)$
            $K^+ = K^+ \cup Y_j$
            f = истинна
        Всё_если
    Всё_цикл
    Если не (f) то
        Выход из цикла
    Всё_если
Всё_цикл

Рассмотрим применение этого алгоритма на практике: пусть дано отношение R, A={a,b,c,d,e,f} – множество атрибутов отношения R, и множество ФЗ S={{a} → {b, c}, {e} → {c, f}, {b} → {e}, {c, d} → {e, f}}. Требуется проверить, является ли суперключом {a, b}.

$\{a, b\}^+ = \{a, b\}$
Цикл по элементам множества S (итерация 1):
а) $\{a\} \rightarrow \{b, c\}$ $\{a\} \rightarrow \{a,b\}^+ \Rightarrow \{a, b\}^+ = \{a, b\} \rightarrow \{b, c\} = \{a, b, c\}$;
б) $\{e\} \rightarrow \{c, f\}$ $\{e\} \not\subset \{a, b\}^+ \Rightarrow \{a, b\}^+ = \{a, b, c\}$;
в) $\{b\} \rightarrow \{e\}$ $\{b\} \rightarrow \{a,b\}^+ \Rightarrow \{a, b\}^+ = \{a, b, c\} \cup \{e\} = \{a, b, c, e\}$;
г) $\{c, d\} \rightarrow \{e, f\}$ $\{c, d\} \not\subset \{a, b\}^+ \Rightarrow \{a, b\}^+ = \{a, b, c, e\}$.
Цикл по элементам множества S (итерация 2):
а) $\{a, b\}^+ = \{a, b, c, e\}$;
б) $\{e\} \rightarrow \{c, f\}$ $\{e\} \subset \{a, b\}^+ \Rightarrow \{a, b\}^+ = \{a, b, c, e\} \cup \{c, f\} = \{a, b, c, e, f\}$.

Далее $\{a,b\}^+$ не изменится, а значит $\{a,b\}^+ = \{a, b, c, e, f\}$. Множество $\{a,b\}^+ \neq A$, а значит, {a, b} не является ключом и, таким образом не является ни потенциальным ключом, ни суперключом.

Итак, используя приведенный выше алгоритм, становится возможным определить множество потенциальных ключей в отношении. Первичный ключ в отношении выбирается среди потенциальных ключей как наиболее простой в структурном смысле и иногда также называется основным. При этом между составным и простым потенциальными ключами, как правило, выбирают простой, а при выборе между ключом с атрибутами числового типа и ключом с атрибутами строкового типа отдают предпочтения первому.

Различают первичные ключи двух видов: *естественные первичные ключи* – это ключи, которые состоят из атрибутов отношения, описывающих реальные свойства объекта, и *суррогатные ключи* – это ключи, включенные в схему отношения искусственным образом (например, поле идентификатор объекта).

Представленный выше алгоритм, помимо применения для определения множества потенциальных ключей отношения, имеет также важное *следствие:* пусть для некоторого отношения R с множеством атрибутов A определено множество ФЗ $S=\{X_1 \rightarrow Y_1,\ X_2 \rightarrow Y_2, \ldots\}$. Любая ФЗ $\{X \rightarrow Y\}$ (где $X \subset A, Y \subset A$) следует из множества S (т.е. $\{X \rightarrow Y\} \in S^+$) тогда и только тогда, когда справедливо $Y \subset X^+$.

В действительности, так как каждый атрибут множества X^+ функционально зависим от X (т.е. $X \rightarrow X^+$) исходя из алгоритма построения замыкания множества атрибутов представленного выше, то ФЗ $X \rightarrow Y$ следует из S только в том случае, когда $Y \subset X^+$.

2.2.4. Внешний ключ

Внешний ключ – это один из видов ограничений целостности, определяющий связи между кортежами в реляционной базе данных. Пусть R_1 и R_2 – это реляционные отношения (не обязательно различные), при этом A_1 – подмножество множества атрибутов отношения R_1, которое является потенциальным ключом этого отношения (англ. CK – candidate key). A_2 – *внешний ключ* (англ. FK – foreign key) в отношении R_2, ссылающийся на отношение R_1 тогда и только тогда, когда A_2 является таким подмножеством множества атрибутов отношения R_2, что каждое его значение в отношении R_2 либо является пустым, либо соответствует любому значению потенциального ключа A_1 в отношении R_1.

Из определения логически следуют требования к структуре и значениям внешнего ключа, применяемые в реляционных базах данных:

- если потенциальный ключ составной, то внешний ключ, ссылающийся на него также составной; если потенциальный ключ простой, то внешний ключ также простой;
- домены соответствующих атрибутов внешнего и потенциального ключей должны совпадать;
- для внешнего ключа не требуется, чтобы он был подмножеством первичного ключа отношения;
- значение внешнего ключа в некотором смысле является ссылкой на кортеж, содержащий соответствующее значение потенциального ключа (иногда его называют ссылочным или целевым кортежем), поэтому контроль значений внешних ключей называется *проблемой ссылочной целостности*. *Ссылочное ограничение* – это ограничение, по которому каждому значению внешнего ключа должно соответствовать значение потенциального ключа;
- отношение, содержащее внешний ключ, называется *ссылающимся отношением*, а отношение, содержащее потенциальный ключ, – *целевым отношением*, при этом одно и то же отношение может быть одновременно и ссылочным, и ссылающимся;
- отношение может ссылаться само на себя, кроме того ссылки (через внешние ключи) могут образовывать цикл.

2.2.5. Механизм нормализации и нормальные формы

Процесс проектирования схемы БД в реляционной модели основан на механизме *нормализации отношений*, т.е. преобразовании исходного отношения по определенным в теории нормализации правилам и получение таких (производных) отношений, объединение которых эквивалентно исходному отношению.

Нормальные формы имеют номера. Первые три нормальные формы являются базовыми и предложены Коддом. Позднее Бойсом было дополнено определение третьей нормальной формы (3НФ) и введена нормальная форма Бойса-Кодда (НФБК). Затем были также предложены 4НФ, 5НФ и 6НФ. Нахождение отношения в нормальной форме более высокого порядка более предпочтительно, с точки зрения теории нормализации.

Основным механизмом нормализации является *декомпозиция отношения без потерь информации:* разбиение исходного отношения на две или более проекций, объединение которых должно дать исходное отношение.

Рассмотрим пример выполнения декомпозиции. Пусть дано отношение R:

R:

Код	Статус	Город
3	30	Казань
5	30	Новгород

Применив декомпозицию, отношение можно разбить двумя способами:

1) R_1^1

Код	Статус
3	30
5	30

R_1^2

Код	Город
3	Казань
5	Новгород

2) R_2^1

Код	Статус
3	30
5	30

R_2^2

Статус	Город
30	Казань
30	Новгород

Очевидно, что второй вариант декомпозиции недопустим, так как R_2^1 JOIN $R_2^2 \neq R$. Формальные условия, определяющие допустимость разбиения, задает *теорема Хеза:*

Пусть R – отношение с атрибутами {A, B, C}. Если при этом A – является потенциальным ключом ($A \rightarrow B$ и $A \rightarrow C$), то R равно соединению его проекций {A, B} и {A, C}, т.е. $п_{A,B}(R)$ JOIN $п_{A,C}(R) = R$.

Следуя теореме Хеза при определении производных отношений, можно выполнить декомпозицию без потери информации. Вернувшись к примеру, рассмотренному в начале параграфа, необходимо отметить, что при втором варианте разбиения, очевидно, теряется ФЗ {код}→{город}, что и является в итоге причиной возникновения противоречий.

Далее при рассмотрении нормальных форм для простоты изложения предполагаем, что каждое отношение содержит только один потенциальный ключ, который также является первичным.

Неключевой атрибут – это атрибут, который не входит в первичный ключ рассматриваемого отношения.

Два или несколько атрибутов, образующих множество A, называются *взаимно-независимыми*, если ни один из них не зависит функционально от любого подмножества остальных атрибутов множества A [8].

Все отношения в реляционной модели должны быть нормализованы, т.е. должны находиться хотя бы в первой нормальной форме (1НФ). Отношение находится в *первой нормальной форме* тогда и только тогда, когда все домены его атрибутов содержат только скалярные значения, т.е. значения, являющиеся неделимыми с точки зрения предметной области.

В качестве примера рассмотрим таблицу с информацией о сотрудниках с полями: ФИО, должность, оклад, телефон, отдел, а также информация о семьях сотрудников (имена детей и год их рождения, а также ФИО 2-го родителя).

R:

ID	ФИО	Должность	Оклад	Телефон	Отдел	Семьи
1.	Иванов Иван Иванович	Директор	100	24-24	Директор	Петрова Анна Ивановна: Оля (1990), Маша (1992)
2.	Петров Петр Петрович	Гл. инженер	100	25-25	Отдел главного инженера	Иванова Инна Ивановна: Сережа (1989), Катя (1991); Сидорова Ольга Петровна: Коля (1991)
3.	Сидорова Ольга Петровна	Инженер	50	25-25	Отдел главного инженера	Петров Петр Петрович: Коля (1991)

Можно выделить следующие ФЗ: S = { {ID} → {ФИО}, {ID} → {Должность}, {ID} → {Оклад}, {ID} → {Отдел}, {Отдел} → {Телефон}, {ID} → {Семьи} }. Значения атрибута «Семьи», очевидно, не являются скалярными для обоих случаев, т.е. могут быть разбиты на несколько значений:

R_1:

ID	ФИО	Должность	Оклад	Телефон	Отдел	2-й родитель	**Имя**	Год рожд.
1	Иванов Иван Иванович	Директор	100	24-24	Директор	Петрова Анна Ивановна	Оля	1990
1.	Иванов Иван Иванович	Директор	100	24-24	Директор	Петрова Анна Ивановна	Маша	1992
2.	Петров Петр Петрович	Гл. инженер	100	25-25	Отдел главного инженера	Иванова Инна Ивановна	Сережа	1989
2.	Петров Петр Петрович	Гл. инженер	100	25-25	Отдел главного инженера	Иванова Инна Ивановна	Катя	1991
2.	Петров Петр Петрович	Гл. инженер	100	25-25	Отдел главного инженера	Сидорова Ольга Петровна	Коля	1991
3.	Сидорова Ольга Петровна	Инженер	50	25-25	Отдел главного инженера	Петров Петр Петрович	Коля	1991

Отношение R_1 является нормализованным и находится в первой нормальной форме. Предположим, что у одного человека не может быть двух детей с одинаковым именем, а также, что один человек может одновременно работать только в одном отделе. В этом случае неприводимое множество ФЗ для отношения R_1 примет следующий вид: S_1 = {{ID} → {ФИО}, {ID} → {Должность}, {ID} → {Оклад}, {ID} → {Отдел}, {Отдел} → {Телефон}, {ID, Имя} → {Год рождения}, {Имя, Год рождения} → {2-й родитель}}, при чем {ID, Имя} – первичный ключ.

Очевидно, что отношение R_1 обладает избыточностью: для каждой записи содержится информация о ФИО сотрудника, его должности и окладе, а также повторяется информация о ФИО 2-го родителя, названиях отделов и их телефонах. В связи с этим возникают следующие проблемы, которые иногда также называют *аномалиями* [13]:

- аномалия обновления – в случае изменения должности сотрудника потребуется изменение нескольких кортежей в отношении;

- аномалия удаления – в случае удаления всех записей о сотрудниках отдела будет удалена информация о телефоне отдела;
- аномалия вставки – невозможно добавить информацию об отделе без указания сотрудников отдела.

Физический смысл данных аномалий в том, что отношение R_1 описывает не одну сущность предметной области, а несколько: сотрудники, семьи, отделы. Для того чтобы избавиться от этих противоречий, проведем нормализацию отношения до 2НФ.

Отношение находится во *второй нормальной форме* тогда и только тогда, когда оно находится в первой нормальной форме и каждый его неключевой атрибут неприводимо зависит от первичного ключа (т.е. **не зависит** от части ключа).

Отношение R_1, очевидно, не находится в 2НФ, так как его неключевые атрибуты приводимо зависят от первичного ключа (т.е. **зависят** от части ключа). Проведем декомпозицию следующим образом:

R_2^1:

ID	ФИО	Должность	Оклад
1	Иванов Иван Иванович	Директор	100
2.	Петров Петр Петрович	Гл. инженер	100
3.	Сидорова Ольга Петровна	Инженер	50

R_2^2:

ID	**Имя**	Год рождения
1	Оля	1990
1.	Маша	1992
2.	Сережа	1989
2.	Катя	1991
2.	Коля	1991
3.	Коля	1991

R_2^4:

ID	Отдел	Телефон
1.	Директор	24-24
2.	Отдел главного инженера	25-25
3.	Отдел главного инженера	25-25

R_2^3:

2-й родитель	**Имя**	**Год рождения**
Петрова Анна Ивановна	Оля	1990
Петрова Анна Ивановна	Маша	1992
Иванова Инна Ивановна	Сережа	1989
Иванова Инна Ивановна	Катя	1991
Петров Петр Петрович	Коля	1991

Отношения R_2^1, R_2^2, R_2^3, R_2^4 очевидно находятся во второй нормальной форме, так как либо первичный ключ является простым (R_2^1, R_2^4), либо единственный неключевой атрибут зависит неприводимо от первичного ключа (R_2^2, R_2^3).

В общем виде *правило для выполнения нормализации до 2НФ* выглядит следующим образом.

Пусть имеется отношение R = {A, B, C, D}, в котором {A, B} – первичный ключ. Неприводимое множество ФЗ S = {{A, B}→{C}, {A}→{D}}, т.е. как минимум один атрибут в отношении R (в данном случае атрибут D) приводимо зависит от первичного ключа. Тогда R должно быть заменено на два отношения:

- $R_1 = п_{A,B,C}(R)$, где {A, B} – первичный ключ, {A} – внешний ключ;
- $R_2 = п_{A,D}(R)$, где {A} – первичный ключ.

После применения этого правила для отношения R_1 были получены производные отношения, находящиеся во второй нормальной форме. Однако, очевидно, в отношении R_2^4 сохраняются некоторые из аномалий:

- аномалия добавления – нельзя добавить отдел, в котором не работает ни один сотрудник;
- аномалия удаления – удалив информацию обо всех сотрудниках отдела, удалим информацию о его названии и телефоне.

Физический смысл противоречия остался прежним: в отношении R_2^4 описаны две сущности (Отделы, Сотрудники в отделах).

Отношение находится в *третьей нормальной форме* тогда и только тогда, когда оно находится во второй нормальной форме и ни один неключевой атрибут не находится в транзитивной зависимости от первичного ключа.

В отношении R_2^4 имеет место транзитивная зависимость неключевого атрибута «Телефон» от первичного ключа «ID»: S_2^4 = {{ID} → {Отдел}, {Отдел} → {Телефон}}. Исключение транзитивных зависимостей достигается путем декомпозиции отношения R_2^4.

R_3^1:

ID	Отдел
1	Директор
2.	Отдел главного инженера
3.	Отдел главного инженера

R_3^2:

Отдел	Телефон
Директор	24-24
Отдел главного инженера	25-25

В отношении R_3^1: {ID} – первичный ключ, {Отдел} – внешний ключ. В отношении R_3^2: {Отдел} – первичный ключ.

В общем виде *правило для выполнения нормализации до 3НФ* выглядит следующим образом:

Пусть имеется отношение R = {A, B, C}, в котором {A} – первичный ключ. Неприводимое множество ФЗ S = {{A}→{B},{B}→{C}}, т.е. имеет место транзитивная зависимость неключевого атрибута C от первичного ключа A. Тогда R должно быть заменено на два отношения:

- $R_1 = \pi_{A,B}$ (R), где {A } – первичный ключ, {B} – внешний ключ;
- $R_2 = \pi_{B,C}$ (R), где {B} – первичный ключ.

Нормализация отношений до 3НФ, как правило, является достаточной для исключения большинства аномалий из тех, которые устраняются с использованием нормализации. Однако в некоторых случаях может потребоваться нормализация отношений до нормальных форм более высоких порядков (на практике такие ситуации возникают достаточно редко). Так до сих пор рассматривались отношения с единственным потенциальным ключом. Пусть отношение R имеет минимум два составных потенциальных ключа и при этом эти ключи перекрываются (т.е. часть атрибутов принадлежит обоим ключам). Отношение R находится в *нормальной форме Бойса-Кодда* тогда и только тогда, когда каждая нетривиальная и неприводимая слева ФЗ обладает потенциальным ключом в качестве детерминанта. Менее формальное определение: отношение находится в НФБК тогда и только тогда, когда все детерминанты в неприводимом множестве ФЗ являются потенциальными ключами. Необходимо также заметить, что если условия, определенные нами для отношения R по количеству и структуре потенциальных ключей не выполняются, то НФБК эквивалентна 3НФ.

Рассмотрим отношение B = {Код, Имя_поставщика, Товар, Количество}, которое содержит информацию о том какие поставщики, какой товар и в каком объеме поставляли. При этом, как «Код», так и «Имя_поставщика» являются уникальными. Множество ФЗ отношения S = {{Код} → {Имя_поставщика}, {Имя_поставщика} → {Код}, {Код, Товар} → {Количество}, {Имя_поставщика, Товар} → {Количество}}. Нетрудно заметить, что в отношении имеются два

составных потенциальных ключа: {Код, Товар} и {Имя_поставщика, Товар}, которые частично перекрываются по атрибуту «Товар». Возможное значение переменной отношения представлено ниже.

B:

Код	Имя поставщика	**Товар**	Количество
1	Иванов	Молоко	300
1	Иванов	Хлеб	200
1	Иванов	Кефир	400
1	Иванов	Ряженка	200

Отношение B *находится в 1НФ,* так как, очевидно, все домены содержат только скалярные значения.

Отношение B *находится в 2НФ,* так как оно находится в 1НФ и единственный неключевой атрибут «Количество», неприводимо зависит от потенциальных ключей.

Отношение B *находится в 3НФ,* так как оно находится в 2НФ и отсутствуют транзитивные зависимости.

Отношение B *не находится в НФБК,* так как не все детерминанты являются потенциальными ключами (например, {Код} в ФЗ {Код} → {Имя_поставщика}).

Очевидно, что отношение B обладает избыточностью, хотя и находится в 3НФ: информация об имени поставщика, которое определяется уникальным образом его кодом, повторяется несколько раз, что приводит к возникновению аномалии обновления (при изменении имени поставщика потребуется обновление нескольких кортежей в отношении). Для исключения избыточной информации можем разбить исходное отношение B на две проекции:

B_1:

Код	Имя поставщика
1	Иванов

B_2:

Код	**Товар**	Количество
1	Молоко	300
1	Хлеб	200
1	Кефир	400
1	Ряженка	200

Оба отношения B_1 и B_2, очевидно, находятся в НФБК, так как они находятся в 3НФ и в каждом из них только один потенциальный ключ.

Для рассмотрения четвертой нормальной формы требуется ввести определение *многозначных зависимостей*. Пусть A, B и C яв-

ляются произвольными подмножествами множества атрибутов отношения R. Говорят, что B многозначно зависит от A (A–››B) тогда и только тогда, когда множество значений атрибутов B зависит от множества значений атрибутов A и не зависит от множества значений атрибутов C.

Рассмотрим ненормализованное отношение с информацией о преподавателях и читаемых ими курсах.

Курс	Преподаватель	Учебники
Физика	Иванов Петров	Основы механики Оптика
Математика	Иванов	Основы механики Векторный анализ Тригонометрия

В данном отношении определено, какие курсы могут читать преподаватели, а также какие учебники эти преподаватели могут использовать. Проведя нормализацию до 1НФ (т.е. перейдя к доменам, содержащим скалярные значения), получим следующее отношение:

M:

Курс	**Преподаватель**	**Учебники**
Физика	Иванов	Основы механики
Физика	Иванов	Оптика
Физика	Петров	Основы механики
Физика	Петров	Оптика
Математика	Иванов	Основы механики
Математика	Иванов	Векторный анализ
Математика	Иванов	Тригонометрия

Таким образом, встречаются все возможные комбинации преподавателей с учебниками для различных курсов. При этом любой кортеж r = (c, t, b) может быть включен в отношение M в том случае, если курс с читается преподавателем t по книге b. Тогда справедливо следующее ограничение: если в отношении M есть два кортежа $r_1 = (c, t_1, b_1)$ и $r_2 = (c, t_2, b_2)$, то должны присутствовать также кортежи $r_3 = (c, t_1, b_2)$ и $r_4 = (c, t_2, b_1)$, т.е.:

$$((c, t_1, b_1) \in M \wedge (c, t_2, b_2) \in M) \Leftrightarrow ((c, t_1, b_2) \in M \wedge (c, t_2, b_1) \in M).$$

Первичный ключ в данном отношении – {Курс, Преподаватель, Учебники}, т.е. в отношении имеется только один ключ и не

представлено ни одного неключевого атрибута, значит, отношение находится в НФБК. Несмотря на этот факт данное отношение, очевидно, обладает большой избыточностью.

Чтобы решить проблему избыточности, требуется выполнить декомпозицию. Понять способ выполнения декомпозиции в данном случае поможет определение многозначных зависимостей, представленное ранее. Действительно, каждому значению атрибута «Курс» соответствует множество значений атрибута «Преподаватель» (например, физику читают Иванов и Петров) и при этом значения атрибута «Преподаватель» не зависят от значений атрибута «Учебники». Таким образом, имеет место многозначная зависимость: {Курс}–››{Преподаватель}.

При этом каждому значению атрибута «Курс» соответствует множество значений атрибута «Учебники», которые не зависят от значений атрибута «Преподаватель». Последний факт имеет место всегда, т.е. многозначные зависимости всегда существуют парами. Таким образом, в отношении M имеет место многозначная зависимость: {Курс}–››{Преподаватель}|{Учебники}.

Очевидно, что если множество значений атрибута в зависимой части многозначных зависимостей состоит из одного атрибута, то многозначная зависимость является функциональной. Таким образом, ФЗ является частным случаем многозначных зависимостей.

Способ выполнения декомпозиции исходного отношения M, рассмотренного выше, определяет *теорема Фейгина,* являющаяся обобщением представленной ранее теоремы Хеза.

Пусть A, B и C являются множествами атрибутов отношения R = {A, B, C}. Отношение R будет равно соединению его проекций $\pi_{A,B}(R)$ и $\pi_{A,C}(R)$ тогда и только тогда, когда для отношения R выполняется многозначная зависимость A–››B|C.

Таким образом, исходное отношение M необходимо разбить на два отношения следующим образом:

M_1:

Курс	**Преподаватель**
Физика	Иванов
Физика	Петров
Математика	Иванов

M_2:

Курс	**Учебники**
Физика	Основы механики
Физика	Оптика
Математика	Основы механики
Математика	Векторный анализ
Математика	Тригонометрия

В отношениях M_1 и M_2:

- первичные ключи составные;
- неключевые атрибуты отсутствуют;
- атрибут «Курс» является внешним ключом.

Таким образом, отношения M_1 и M_2 также находятся в НФБК, но в них отсутствуют многозначные зависимости и, очевидно, аномалии, присущие отношению М.

Отношение находится *в четвертой нормальной форме* (4НФ) тогда и только тогда, когда оно находится в НФБК и все многозначные зависимости являются фактически функциональными зависимостями. Более формальное определение звучит следующим образом: *отношение R находится в четвертой нормальной форме* (4НФ) тогда и только тогда, когда если существуют такие подмножества А и В атрибутов отношения R, что выполняется нетривиальная многозначная зависимость A–››B, то все атрибуты отношения R также функционально зависимы от атрибута А. В примере, представленном выше, отношение М – не находится в 4НФ, тогда как полученные на его базе отношения M_1 и M_2 – находятся в 4НФ.

Рассмотрим еще один пример отношения. Пусть отношение К представляет информацию о том, какие поставщики, какие детали в рамках какого проекта поставляют. Возможное значение переменной отношения К представлено ниже.

К:

Поставщик	**Детали**	**Проекты**
Иванов	1	2
Иванов	2	1
Петров	1	1
Иванов	1	1

Отношение К является *полностью ключевым* (т.е. первичный ключ – совокупность всех атрибутов отношения), не содержит нетривиальных и многозначных зависимостей, а, следовательно, находится в 4НФ. Рассмотрим декомпозицию отношения К на три проекции:

K_1

Поставщик	Детали
Иванов	1
Иванов	2
Петров	1

K_2

Детали	Проекты
1	2
2	1
1	1

K_3

Проекты	Поставщик
2	Иванов
1	Иванов
1	Петров

Проверим допустимость такого разбиения. Для этого последовательно выполним композицию отношения K_1, K_2 и K_3:

$K_4 = K_1$ JOIN K_2:

Поставщик	Детали	Проекты
Иванов	1	2
Иванов	1	1
Иванов	2	1
Петров	1	2
Петров	1	1

◄ линейный кортеж

В отношении K_4 жирным шрифтом выделен кортеж, которого не было в отношении K (такой кортеж называют *линейным к отношению K*). Нетрудно проверить возникновение линейного кортежа в случае соединения отношений K_1 и K_3, а также K_2 и K_3.

Однако требуется закончить композицию проекций – выполним соединение отношения K_4 и отношения K_3 по атрибутам {Проекты, Поставщики}.

$K_5 = K_4$ JOIN K_3:

Поставщик	Детали	Проекты
Иванов	1	2
Иванов	2	1
Петров	1	1
Иванов	1	1

Полученное отношение K_5 эквивалентно исходному отношению K. Таким образом, отношение K обладает следующим свойством: его нельзя разбить ни на какие две проекции, которые после соединения будут ему эквиваленты, но при этом можно разбить на три проекции, которые после соединения будут ему эквивалентны. Это свойство называют свойством *трехдекомпозируемости* (записывают: 3-декомпозируемость).

Пусть R – переменная отношения, в которой A, B, … Z – некоторые подмножества множества ее атрибутов. Если декомпозиция любого допустимого значения переменной R на отношения, состоящие из множеств атрибутов A, B, … Z, является декомпозицией без потерь, то говорят, что отношение R удовлетворяет *зависимости соединения* *{A, B, … Z}. Иными словами, отношение R удовлетворяет зависимости соединения *{A, B, … Z} тогда и только тогда, когда оно эквивалентно соединению своих проекций по подмножествам A, B, … Z множества атрибутов. *Зависимость соединения* является обобще-

нием понятия «многозначная зависимость» и, как следствие, понятия «функциональная зависимость».

Количество подмножеств, входящих в состав зависимости соединения, определяет степень декомпозируемости отношения (для n>2 записывают n-декомпозируемость, где n – представляет степень декомпозируемости).

Зависимость соединения *{A, B, … Z} является *тривиальной* тогда и только тогда, когда по крайней мере одно из подмножеств A, B, … Z является множеством всех атрибутов отношения. Отношение находится в *пятой нормальной форме* тогда и только тогда, когда каждая нетривиальная зависимость соединения в нем определяется потенциальным ключом этого отношения.

Рассмотренное выше отношение K не находится в 5НФ, так как имеет место нетривиальная зависимость соединения *{{Поставщик, Детали}, {Детали, Проекты}, {Поставщики, Проекты}}, при этом ни одно из подмножеств не является потенциальным ключом (т.е. отношение K является полностью ключевым). Приведение отношения к 5НФ заключается в его декомпозиции на три отношения, представленных выше: K_1, K_2, K_3.

На первый взгляд уровень нормализации исходного отношения K является достаточным для исключения всех возможных аномалий. Однако для данного отношения практически невозможно автоматически (на уровне СУБД) контролировать ограничение: каждый поставщик имеет в своем ассортименте ограниченный список деталей и участвует в ограниченном списке проектов, к тому же каждому проекту нужен ограниченный список деталей. Таким образом, при разработке приложений контроль этого ограничения (так называемого *циклического ограничения* [13]) должно быть реализовано в рамках приложения, а значит, не может контролироваться СУБД. Все это потенциально может приводить к нарушению логической целостности данных.

Пятая нормальная форма является окончательной в контексте операций проекции и соединения, т.е. отношение, находящееся в 5НФ, находится во всех формах других нормальных формах низших порядков [12]. Однако имеется обобщение 5НФ, применяемое в *хронологических* (иногда *темпоральных*) *базах данных*.

Хронологическая (или *темпоральная*) *база данных* – это база данных, содержащая историю изменения постоянных данных, т.е. данные, относящиеся к прошлым и, возможно, к будущим периодам

времени, тогда как обычная, нехронологическая БД содержит лишь текущую версию данных.

Итак, переменная отношения находится в *шестой нормальной форме* (6НФ) тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Не вдаваясь в особенности функционирования темпоральных БД, рассмотрим пример, демонстрирующий необходимость выполнения «нормализации до конца», т.е. до 6НФ [23].

Пусть имеется темпоральная БД, в которой отношение T представляет информацию об адресах работников, а также об их должностях. При этом, так как БД является хронологической, то все изменения данных должны быть сохранены. Рассмотрим возможное значение переменной отношения T.

T:

ID	Должность	Домашний адрес	**Интервал**
1	Слесарь	ул. Ленина, 10	[01-01-2000:10-02-2003]
1	Слесарь	ул. Советская, 22	[11-02-2003:15-06-2006]
1	Бригадир	ул. Советская, 22	[16-06-2006:05-03-2009]

Значение атрибута «Интервал» определяет, в какой момент времени значения атрибутов в кортеже были актуальными для сотрудника. Например, по представленному значению переменной T можем сказать, что до 15.06.2006 сотрудник с идентификатором 1 работал слесарем, после чего был переведен на должность бригадира.

Отношение T обладает аномалиями несмотря на то, что, очевидно, находится в 5НФ: независимо от того, какой из неключевых атрибутов («Должность» или «Домашний адрес») будет изменен, в отношение T потребуется добавлять копию всего кортежа. Так, при смене адреса проживания (1-й и 2-й кортежи) должность сотрудника не меняется. Соответственно при смене должности (2-й и 3-й кортежи) не меняется уже домашний адрес сотрудника.

Преодоление такого рода избыточности и переход к 6НФ осуществляется декомпозицией отношения:

T_1:

ID	Должность	**Интервал**
1	Слесарь	[01-01-2000: 15-06-2006]
1	Бригадир	[16-06-2006: 05-03-2009]

T_2:

ID	Домашний адрес	**Интервал**
1	ул. Ленина, 10	[01-01-2000: 10-02-2003]
1	ул. Советская, 22	[11-02-2003: 05-03-2009]

Полученные в результате декомпозиции отношения T_1 и T_2 находятся в шестой нормальной форме и фактически лишены любых видов аномалий, таким образом, процесс нормализации выполнен до конца [8].

2.3. Основы использования языка SQL

Исходя из требований к реляционным СУБД (5-е и 7-е правила Кодда), каждая такая СУБД должна поддерживать высокоуровневый реляционный язык для обеспечения взаимодействия пользователей с данными в БД. Сегодня де-факто для большинства реляционных СУБД таким языком является структурированный язык запросов (англ. SQL – Structured Query Language), о чем также свидетельствует тот факт, что множество альтернативных подходов объединяются сегодня теоретиками БД в рамки общей концепции NoSQL (англ. not only SQL). В данном разделе рассмотрим основные операторы языка SQL, его структуру и применение.

2.3.1. Структура языка SQL

Язык SQL является универсальным декларативным компьютерным языком, предназначенным для взаимодействия пользователей с реляционной базой данных. Особенностью декларативных языков является то, что вместо написания алгоритма решения задачи некоторым образом описывается то, что требуется получить в качестве результата.

Язык SQL разрабатывался с целью стандартизовать язык взаимодействия с реляционными СУБД. Первый стандарт языка SQL1 был разработан в 1986г. К настоящему моменту известны следующие стандарты: SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008. Однако, несмотря на такой набор стандартов, далеко не все современные СУБД поддерживают одинаковые диалекты языка. Связано это в первую очередь с тем, что вопрос стандартизации тех или иных компонентов языка поднимался уже после того, как разработчики СУБД объявляли об их поддержке. На текущий момент все усилия по проверке СУБД на соответствие языка SQL текущей версии стандарта ложатся на ее разработчика.

В стандарте SQL-92 определены четыре уровня соответствия конкретной реализации стандарту: базовый, переходный, промежуточный, полный. Однако стандарт SQL:1999 «отошел» от этой практики и имеет модульную структуру. Основная часть стандарта вынесена в отдельный раздел (SQL/Foundation), остальные части составляют отдельные модули. В связи с этим все СУБД, поддерживающие основную часть стандарта SQL:1999, имеют уровень совместимости

Core (единственный возможный), а поддержка остальных модулей предоставлена разработчикам СУБД.

Язык SQL включает в свой состав операторы, инструкции и вычисляемые функции. До появления SQL СУБД поддерживали языки, выполняющие отдельные функции по обработке и управлению данными. Таким образом, для определения схемы БД использовались подъязыки определения данных, для работы с ними – подъязыки модификации данных, а для выборки – подъязык запросов. Язык SQL вобрал в себя функции всех подъязыков (являясь, таким образом, универсальным) для взаимодействия с СУБД, поэтому принято различные операторы языка SQL разделять на следующие группы:

- *подъязык запросов* состоит из оператора SELECT, предоставляющего большой набор опций по разработке всевозможных запросов на выборку данных;
- *подъязык определения данных* (англ. Data Definition Language – DDL) или *подъязык определения схемы БД* (англ. Schema Definition Language – SDL) – группа операторов, предназначенная для создания/изменения/удаления объектов БД;
- *подъязык манипулирования данными* (англ. Data Manipulation Language – DML) – группа операторов, для добавления/изменения/удаления данных в БД;
- *подъязык определения доступа к данным* (англ. Data Control Language – DCL) – группа операторов предназначенных для предоставления/отзыва разрешений к объектам БД различным пользователям;
- *подъязык управления транзакциями* (англ. Transaction Control Language – TCL) – группа операторов для управления состоянием транзакции.

Операторы этих групп будут рассмотрены далее, здесь же необходимо отметить, что, так как SQL не предоставляет возможности написания процедур, а также использования ветвлений и циклов, то различными производителями в рамках их программных продуктов предлагались расширения языка SQL. Наиболее известные и активно используемые из них:

- Transact-SQL (СУБД MS SQL Server);
- PL/SQL (СУБД Oracle);
- PL/pgSQL (СУБД PostgreSQL).

2.3.2. Оператор подъязыка запросов

Как было сказано выше, подъязык запросов в языке SQL состоит из одного оператора, однако, этот оператор обладает большим набором необязательных параметров. Синтаксис оператора SELECT, используемый в СУБД MS Access, представлен ниже [**19**].

```
SELECT [ALL | DISTINCT | DISTINCTROW | TOP]
{* | <таблица>.* | [<таблица>.]<поле1> [AS <псевдоним1>] [, [<таблица>.]<поле2>
[AS <псевдоним2>][, …]]}
[ FROM {<таблица> <псевдоним таблицы>[, …]} ]
[ WHERE <условия отбора> ]
[ GROUP BY <столбец> ]
[ HAVING <условие отбора> ]
[ ORDER BY <список столбцов>];
```

Рассмотрим особенности представленного синтаксиса. Запрос начинается с ключевого слова SELECT, которое говорит о том, что будет произведена выборка (селекция) данных из реляционной БД. После этого может быть указано одно из ключевых слов:

- ALL – указывает, что необходимо вывести все (в том числе повторяющиеся) записи; опция используется по умолчанию;
- DISTINCT – исключает из результирующей выборки все записи, которые содержат одинаковые значения в выбранных полях;
- DISTINCTROW – исключает из результирующей выборки только записи, которые содержат одинаковые значения в выбранных полях и при этом были получены в результате объединения таблиц (оператор используется в MS Access и MS SQL Server);
- TOP – указывает максимальное число выбираемых строк (оператор используется в MS Access и MS SQL Server). В других СУБД (PostgreSQL, MySQL) для указания максимального числа выводимых строк используется ключевое слово LIMIT.

После указанных операторов следует либо:

- символ «*» – выбрать все поля из указанных источников;
- константа любого допустимого типа;
- список выводимых полей с указанием таблиц, из которых эти поля выбираются.

Инструкции, расположенные после ключевого слова SELECT и до ключевого слова FROM, определяют схему результирующей выборки, т.е. то, какие поля, с какими именами и в каком порядке будут представлены (операция проекции).

Источники данных (таблицы, из которых происходит селекция) указываются в необязательной секции после ключевого слова FROM. Хотя секция FROM является необязательной, она присутствует во всех запросах к таблицам БД, так как она, очевидно, может быть опущена только в том случае, если все выводимые поля являются константами.

Следующей необязательной секцией является секция, начинающаяся с ключевого слова WHERE, которая определяет, какие записи (кортежи) должны попасть в результирующую выборку. Для этого после ключевого слова WHERE указывается условие отбора, представляемое предикатом (логическое выражение). В результирующую выборку попадают только те записи, для которых предикат принимает истинное значение, что соответствует операции селекции, рассмотренной ранее. Логическое выражение состоит из условий (предикатов) для различных полей, выбираемых записей, некоторые из которых перечислены ниже:

- предикаты сравнения (=, <>, >, >=, <, <=);
- логические операторы AND и OR – логическое «И» и «ИЛИ»;
- оператор отрицания NOT – отрицание: если предикат без оператора отрицания принимает ложное значение, то с ним – истинное и наоборот;
- предикат Between A and B – принимает значения между A и B: истинен, когда сравниваемое значение попадает в заданный диапазон, включая границы диапазона;
- предикат «вхождения в множество» IN – истинен тогда, когда сравниваемое значение входит в множество заданных значений, при этом множество значений может быть задано простым перечислением или встроенным подзапросом;
- предикаты сравнения с образцом LIKE: предикат истинен, если сравниваемое значение соответствует шаблону, и ложен в противном случае, при этом шаблон может содержать символ «%» («*» для Access) для обозначения любого числа любых символов и символ «_» («?» для Access) для обозначения любого одного символа;
- предикат сравнения с неопределенным значением IS NULL – истинен, когда сравниваемое значение является пустым, при этом необходимо помнить, что сравнение с пустым значением значения некоторого поля с использованием оператора «=» невозможно;

- предикат существования EXIST – применяется во вложенных запросах для определения непустого или пустого множества, являющегося результатом выборки.

После определения условия выборки необходимо также определить условия группировки. Группировка подразумевает определение списка полей, по значениям которых поля объединяются в группы, при этом для каждой полученной группы записей к значениям полей, не участвующих в группировке, могут быть применены так называемые *агрегатные функции* (некоторые из них представлены в табл.7). Список полей для группирования указывается после ключевого слова GROUP BY, причем только эти поля могут быть указаны после ключевого слова SELECT, тогда как к другим выводимым полям должны быть применены агрегатные функции.

Таблица 7

Распространенные агрегатные функции языка SQL

Функция	**Результат**
COUNT	Количество строк или непустых значений полей, которые выбрал запрос
SUM	Сумма всех выбранных значений данного поля
AVG	Среднеарифметическое значение всех выбранных значений данного поля
MIN	Наименьшее из всех выбранных значений данного поля
MAX	Наибольшее из всех выбранных значений данного поля

Важно понимать, что указанное в секции WHERE условие выбора работает для записей до выполнения группировки. Для того чтобы отбросить часть результатов, полученных после группировки используется инструкция HAVING, после которого указывается предикат, составленный по правилам, аналогичным инструкции WHERE.

Заключительной из рассматриваемых секций является секция определения порядка сортировки, в которой после ключевого слова ORDER BY указывается список полей, по значениям которых будет проходить сортировка выводимых результатов, а также порядок сортировки (ASC – по возрастанию, DESC – по убыванию). Необходимо отметить, что хотя большинство секций не являются обязательными, их порядок при формировании запроса не может быть изменен.

Рассмотрим выполнение различных вариантов запроса SELECT на примере следующей БД:

R1 (Группы)

ФИО	Группа
Петров Ф.И.	АИ21
Сидоров К.А.	АИ21
Миронов А.В.	АИ21
Трофимов П.А.	АИ22
Иванова Е.А.	АИ22
Уткина Н.В.	АИ22

R2 (Дисциплины)

Группа	Дисциплина
АИ21	Базы данных
АИ21	Моделирование
АИ22	Сети ЭВМ

R3 (Оценки)

ФИО	Дисциплина	Оценка
Петров Ф.И.	Базы данных	5
Сидоров К.А.	Базы данных	4
Миронов А.В.	Базы данных	2
Петров Ф.И.	Моделирование	5
Сидоров К.А.	Моделирование	4
Миронов А.В.	Моделирование	NULL
Трофимов П.А.	Сети ЭВМ	4
Иванова Е.А.	Сети ЭВМ	5
Уткина Н.В.	Сети ЭВМ	5

Следующий запрос выводит студентов, получивших неудовлетворительные оценки по любой из дисциплин, либо не явившихся на экзамен:

```
SELECT R3.ФИО FROM R3
WHERE (R3.Оценка IS NULL) OR (R3.Оценка < 3)
ORDER BY R3.ФИО;
```

ФИО
Миронов А.В.
Миронов А.В.

Студент «Миронов А.В.» был выведен дважды, потому что в таблице R3 имеется две строки, удовлетворяющие заданному условию отбора, и обе из них в качестве значения атрибута «ФИО» имеют строку «Миронов А.В.». Следующий запрос демонстрирует применение инструкции DISTINCT и выводит название дисциплин (сортировка по убыванию), по которым имеется хотя бы одна отличная оценка:

```
SELECT DISTINCT R3.Дисциплина FROM R3
WHERE R3.Оценка = 5
ORDER BY R3.Дисциплина DESC;
```

Дисциплина
Сети ЭВМ
Моделирование
Базы данных

Следующий запрос вычисляет количество студентов, получивших отличную оценку для каждой из дисциплин, при этом дисциплины, по которым нет ни одной отличной оценки, в результирующую выборку не попадут:

```
SELECT R3.Дисциплина, COUNT(R3.Оценка) AS Количество5 FROM R3
WHERE R3.Оценка = 5
GROUP BY R3.Дисциплина
ORDER BY R3.Дисциплина DESC;
```

Дисциплина	Количество5
Сети ЭВМ	2
Моделирование	1
Базы данных	1

Следующий запрос предназначен для выбора всех оценок студентов «Петров Ф.И.» и «Уткина Н.В.»:

```
SELECT * FROM R3
WHERE R3.ФИО IN ('Петров Ф.И.', 'Уткина Н.В.');
```

ФИО	Дисциплина	Оценка
Петров Ф.И.	Базы данных	5
Петров Ф.И.	Моделирование	5
Уткина Н.В.	Сети ЭВМ	5

Следующий запрос выбирает название всех дисциплин, в которых есть последовательность символов «ан»:

```
SELECT Дисциплина FROM R2
WHERE Дисциплина LIKE '%ан%';
```

Дисциплина
Базы данных
Моделирование

Следующий запрос не содержит источников данных и демонстрирует, что нельзя использовать оператор «=» для сравнивания с пустым значением:

```
SELECT (null is null) as f1, (null = null) as f2;
```

В зависимости от используемой СУБД данный запрос вернет разные результаты, которые указывают, что результат сравнения двух пустых значений на равенство (оператор «=») не определен:

MS Access 2003	
f1	**f2**
-1	NULL

MySQL 5	
F1	**f2**
1	NULL

PostgreSQL 9	
f1	**f2**
t	NULL

На практике гораздо чаще требуется производить выборку сразу из нескольких таблиц БД. Для этого в языке SQL применяются различные виды объединений таблиц (соответствуют реляционной операции соединения отношений). Для простоты изложения будем рассматривать объединение двух абстрактных таблиц A и B. Понятно, что для объединения трех таблиц (A, B и C) требуется выполнить вначале объединение любых двух таблиц (например, A и B), а затем полученный результат объединить с третьей таблицей. Существуют следующие варианты объединения двух таблиц A и B:

1) *перекрестное объединение* – объединение кортежей двух таблиц, в результате которого результирующее множество кортежей составляется из всех возможных вариантов объединения кортежей таблиц A и B (условие объединения не требуется);

2) *внутреннее объединение* – объединение кортежей двух таблиц, в результате которого результирующее множество кортежей составляется из всех возможных вариантов объединения кортежей таблиц A и B, удовлетворяющих условию объединения;

3) *внешнее объединение* бывает трех видов:

- *левое внешнее объединение* – объединение кортежей двух таблиц, в результате которого результирующее множество кортежей составляется из всех возможных вариантов объединения кортежей таблиц A и B, удовлетворяющих условию объединения, а также из всех кортежей таблицы A (дополненных пустыми значениями вместо значений кортежа из B), для которых невозможно найти такие кортежи в таблице B, чтобы выполнялось условие объединения;
- *правое внешнее объединение* – объединение кортежей двух таблиц, в результате которого результирующее множество кортежей составляется из всех возможных вариантов объединения кор-

тежей таблиц A и B, удовлетворяющих условию объединения, а также из всех кортежей таблицы B (дополненных пустыми значениями вместо значений кортежа из A), для которых невозможно найти такие кортежи в таблице A, чтобы выполнялось условие объединения;

- *полное внешнее объединение* – объединение кортежей двух таблиц, в результате которого результирующее множество кортежей составляется из объединения множества кортежей, полученных в результате левого и правого внешних объединений.

Следующие запросы дают одинаковый результат на любых значениях переменных отношений, однако, первый вариант считается устаревшим с точки зрения синтаксиса. В первом случае производится перекрестное объединение с выполнением условия отбора (оно же в данном случае условие соединения), тогда как во втором случае условие отбора указывается специальным образом:

```
1) SELECT R1.ФИО, R2.Дисциплина
FROM R1, R2
WHERE R1.Группа = R2.Группа;
2) SELECT R1.ФИО, R2.Дисциплина
FROM R1 INNER JOIN R2
ON R1.Группа = R2.Группа;
```

ФИО	Дисциплина
Петров Ф.И.	Базы данных
Сидоров К.А.	Базы данных
Миронов А.В.	Базы данных
Петров Ф.И.	Моделирование
Сидоров К.А.	Моделирование
Миронов А.В.	Моделирование
Трофимов П.А.	Сети ЭВМ
Иванова Е.А.	Сети ЭВМ
Уткина Н.В.	Сети ЭВМ

Следующий запрос демонстрирует применение вложенных запросов и левого внешнего объединения: в результирующую выборку попадают все кортежи, полученные в результате внутреннего объединения, и те кортежи из левой таблицы, которые не попали в эту выборку:

SELECT R1.ФИО, R4.Дисциплина
FROM R1 LEFT JOIN (SELECT * FROM R2 WHERE Группа = «АИ21») AS R4 ON R1.Группа = R4.Группа;

ФИО	Дисциплина
Петров Ф.И.	Базы данных
Сидоров К.А.	Базы данных
Миронов А.В.	Базы данных
Петров Ф.И.	Моделирование
Сидоров К.А.	Моделирование
Миронов А.В.	Моделирование
Трофимов П.А.	NULL
Иванова Е.А.	NULL
Уткина Н.В.	NULL

В данном примере вначале выполняется подзапрос, полученное в результате отношение именуется псевдонимом R4 и затем участвует в левом внешнем объединении (с правой стороны) с таблицей R1. Так как группа «АИ22» была исключена условие отбора в подзапросе, то студентам из этой группы в отношении R1 не удалось поставить ни одного кортежа из R4.

Известно три алгоритма для выполнения операции соединения (табл.8):

- *соединение вложенными циклами* (англ. nested loops join – NLJ) для двух таблиц заключается в построении декартова произведения исходных таблиц в результате последовательного циклического перебора записей с анализом заданного условия соединения для каждой из комбинаций записей;
- *соединение слиянием сортированных списков* (англ. sort-merge join – SMJ) для двух таблиц, записи в которых должны быть отсортированы по значениям полей участвующих в соединении, осуществляется за один проход по каждой из входных таблиц, что обеспечивает выигрыш в скорости по сравнению с NLJ;
- *соединение хэшированием* (англ. hash join – HJ) меньшая из двух входных таблиц помещается в структуру данных хэш-таблица, обеспечивающую высокую скорость поиска, после чего для каждой строки из большой таблицы выполняется поиск значений, соответствующих условию соединения.

Таблица 8

Преимущества и недостатки алгоритмов соединения

Алгоритм	Преимущества	Недостатки
NLJ	+ базовый, а значит, незаменимый алгоритм; + самый быстрый алгоритм для получения только одной строки результата (выражение типа EXISTS); + время выполнения алгоритма при увеличении числа записей в объединяемых таблицах растет линейно.	- в общем случае является самым медленным с точки зрения числа необходимых операций.
SMJ	+ эффективнее при условии изначальной сортировки входных таблиц; + может эффективно использоваться при больших размерах соединяемых таблиц.	- необходимость накладных расходов на сортировку входных данных; - менее эффективен с точки зрения использования памяти, вследствие чего на практике в СУБД используется редко.
HJ	+ самый эффективный по числу операций вид соединения при небольшом размере меньшей из двух соединяемых таблиц.	- в условии соединения только равенство; - большая потребность в памяти для хэш-таблицы; - выполнение производится только после полного формирования хэш-таблицы.

В заключение необходимо отметить, что рассмотренные алгоритмы соединения, очевидно, могут быть применены в любых ситуациях, требующих комбинирования данных, содержащихся в нескольких списках.

2.3.3. Операторы подъязыков определения и модификации данных

Обе эти группы операторов, как правило, синтаксически не существенно различаются в различных СУБД и при использовании

стандартных возможностей, запросы с их использованием могут быть достаточно просто перенесены с одной СУБД на другую.

Операторы подъязыка DDL используются с целью определения схемы данных в реляционной базе данных, а именно для создания/модификации/удаления таблиц, представлений, последовательностей, хранимых процедур, триггеров и других объектов, определенных в конкретной СУБД. В данном подъязыке определены следующие операторы, которые могут быть использованы для создания различных объектов БД:

- CREATE – создание объектов;
- ALTER – изменение структуры объекта (например, добавление поля в таблице, изменение типа поля в таблице и т.д.);
- DROP – удаление объекта.

Рассмотрим синтаксис оператора для создания таблицы в БД:

```
CREATE TABLE <имя таблицы> (
<поле1> <тип поля1> [<ограничение1>]
[, <поле2> <тип поля2> [<ограничение2>]]
[, …]
[<список ограничений>]
);
```

Как правило, при создании таблицы с использованием оператора CREATE TABLE требуется указать имя таблицы (возможно, имя схемы, в которой создается таблица), список полей таблицы, их типы, а также в случае необходимости возможно определить ограничения целостности для создаваемых таблиц (например, первичный ключ, внешний ключ, уникальное поле, непустое значение, значение из диапазона и т.д.). Необходимо отметить, что в различных СУБД синтаксис в целом отличается в зависимости от реализуемых этой СУБД дополнительных возможностей. Пример использования операторов подъязыка DDL с комментариями для СУБД PostgreSQL [**6**] представлен ниже.

```
-- удаление таблиц, если они существуют
DROP TABLE IF EXISTS "select", test, test2;
-- создание таблицы с именем select
CREATE TABLE "select" (
"group" numeric(11,0) NOT NULL DEFAULT '0', -- значение по умолчанию 0
"insert" real,
CONSTRAINT select_pkey PRIMARY KEY ("group") -- первичный ключ
);
-- создание таблицы с использованием оператора SELECT
CREATE TABLE test2 AS SELECT * FROM "select";
```

```
-- переименование столбца и изменение его типа
ALTER TABLE test2 RENAME COLUMN "group" TO id;
ALTER TABLE test2 ALTER COLUMN id TYPE int;
-- определение первичного ключа для таблицы test2
ALTER TABLE test2 ADD PRIMARY KEY(id);
-- переименование таблицы select
ALTER TABLE "select" RENAME TO test;
-- добавление ограничения уникальности для поля insert
ALTER TABLE test ADD CONSTRAINT uinsert UNIQUE ("insert");
-- определение значения по умолчанию
ALTER TABLE test ALTER COLUMN "insert" SET DEFAULT 0.123;
```

Операторы подъязыка DML предназначены для модификации данных, хранимых в реляционной СУБД. Данные в СУБД хранятся в таблицах, состоящих из записей, которые в свою очередь состоят из полей. В связи с этим подъязык включает в себя следующие операторы, позволяющие обрабатывать записи и поля:

- INSERT – вставить запись (записи) в таблицу (представление);
- UPDATE – модифицировать существующие данные;
- DELETE – удалить записи.

Операторы UPDATE или DELETE могут за один свой вызов модифицировать сразу любое число записей, содержащихся в таблице. Множество этих записей задается предикатом, который указывается в секции WHERE. Оператор INSERT может быть использован как для вставки одной записи в таблицу, так и для вставки сразу нескольких записей. Рассмотрим общий синтаксис данных операторов.

Для вставки данных в таблицу необходимо использовать оператор INSERT:

```
INSERT INTO <имя_таблицы> [(<список столбцов>)]
VALUES (<список значений>).
```

При этом, как видим, список столбцов можно опустить, но тогда значения из списка значений будут вставлены в соответствии со схемой таблицы.

Для удаления одной, нескольких или всех строк из таблицы применяется оператор DELETE:

```
DELETE FROM <имя_таблицы> [WHERE <условия_отбора>]
```

Сколько записей будет удалено из указанной таблицы, определяет условие отбора. В случае, если секция WHERE опущена, то из таблицы будут удалены все записи.

Запрос на обновление (оператор UPDATE) данных состоит из трех частей, в которых определяется имя обновляемой таблицы, об-

новляемые поля и значения, а также условие отбора записей (последняя секция является необязательной):

```
UPDATE <имя_таблицы>
SET <поле1> = <новое значение1> [,<поле2> = <новое значение2>] [, …]
[ WHERE <условие отбора> ].
```

Пример использования операторов DML с комментариями для СУБД PostgreSQL представлен ниже.

```
-- вставка 3 строк в таблицу test2
INSERT INTO test2 VALUES (1,0.1), (2,0.2), (3,0.3);
-- удаление первичного ключа в таблице test
ALTER TABLE test DROP CONSTRAINT select_pkey;
-- добавление поля с автоувеличением
ALTER TABLE test
ADD COLUMN i1 serial NOT NULL PRIMARY KEY;
-- вставка данных в таблицу test с частичным указанием значений строки
INSERT INTO test("group") VALUES (0) RETURNING *;
-- вставка данных в таблицу test из запроса SELECT
INSERT INTO test("insert","group")
SELECT "insert", id FROM test2;
-- увеличение значений поля insert для всех строк в таблице test на 1
UPDATE test SET "insert" = "insert" + 1;
-- удаление строк из таблицы test, в которых значение поля group < 2
DELETE FROM test WHERE "group" < 2;
-- создание поля с типом перечисление из трех значений
DROP TYPE IF EXISTS etype;
CREATE TYPE etype AS ENUM('a','b','c');
ALTER TABLE test ADD COLUMN e1 etype;
-- установка значения поля e1 для всех записей
UPDATE test SET e1 = 'c' RETURNING *;
-- следующая команда вызовет ошибку, так как `d' не определено среди
-- допустимых значений созданного ранее типа etype
UPDATE test SET e1 = 'd';
```

Не все из представленных выше синтаксических конструкций могут быть перенесены в другие СУБД, поскольку в примерах использовались специфические возможности СУБД PostgreSQL.

Другие операторы. В соответствии с 5-м правилом Кодда реляционный язык должен поддерживать управление объектами БД и транзакциями. Выше были выделены два подъязыка, содержащих операторы, которые реализуют эти функции в рамках языка SQL.

Операторы подъязыка DCL предназначены для осуществления *администрирования ресурсов БД*, т.е. выполнения функций по присвоению или отмене отдельных прав (привилегий) на использование, как всей БД, так и ее объектов (таблиц, записей, полей, пред-

ставлений и т.д.) отдельными пользователями БД и их группами. В рамках подъязыка определены операторы для предоставления пользователю (группе пользователей) разрешения на операции с объектов БД (оператор GRANT) и отзыва разрешений выданных ранее (оператор REVOKE).

Подъязык TCL включает в свой состав (как правило) два оператора, функции которых рассмотрены при описании механизма транзакций: COMMIT - фиксация изменений, выполненных транзакцией и ROLLBACK – откат всех изменений, сделанных в рамках отменяемой транзакции. Операции, включаемые в транзакцию, начинаются с использования специального слова: BEGIN [TRANSACTION] и заканчиваются выполнением оператора COMMIT или ROLLBACK. Любое из завершений транзакций (фиксация или откат) должно освободить все блокировки, которые были установлены завершаемой транзакцией.

2.3.4. Преимущества и недостатки языка SQL

Язык SQL разрабатывался как универсальный язык для взаимодействия с СУБД. Поэтому, несмотря на наличие различных диалектов, в целом можно констатировать, что синтаксис операторов основных групп языка (DDL, DML и язык запросов) практически идентичен в современных реляционных СУБД, что позволяет *не зависеть от конкретной СУБД.*

Последние версии стандарта языка SQL хорошо проработаны и формализованы, имеются тесты для выявления совместимости и соответствия конкретной реализации SQL общепринятым стандартам. *Стандартизация языка* SQL является несомненным его преимуществом.

Декларативность языка SQL позволяет программистам не задумываться о том, каким образом СУБД выбирает и обрабатывает данные.

Вместе с тем, основными теоретиками и разработчиками реляционной модели данных Э. Коддом и К. Дейтом указаны некоторые несоответствия языка SQL, которые *не позволяют считать его полностью реляционным* языком (например, повторяющиеся строки, неопределенные значения, высокая избыточность и т.д.). К тому же, несмотря на то, что SQL задумывался как простой язык, предназначенный в первую очередь для выполнения запросов конечными пользо-

вателями, в конце концов, он *оказался слишком сложным* для них и в настоящее время используется преимущественно прикладными программистами при разработке программного обеспечения.

Наличие большого числа диалектов и отступлений от стандартов в известных СУБД (Oracle, PostgreSQL, MS SQL Server, MySQL и т.д.) создают проблемы с переносимостью в случае использования специфичных операторов и функций.

2.4. Вопросы и задания для самоконтроля

1. Определите основные понятия реляционной модели данных: объект, атрибут, домен, кортеж, отношение, схема отношения.
2. Какие реляционные виды реляционных отношений выделяют?
3. Определите операции реляционной алгебры и приведите примеры их использования?
4. Сформулируйте требования к реляционным БД (правила Кодда)?
5. Функциональные зависимости 1-го и 2-го типов: определения и примеры.
6. Что такое неприводимое множество функциональных зависимостей? Как его получают и для чего используют?
7. Опишите этапы выбора первичного ключа реляционного отношения.
8. Что такое внешний ключ? В чем суть проблемы ссылочной целостности?
9. Нормализация отношения до 3НФ: механизм нормализации, теорема Хеза, определения 1НФ, 2НФ и 3НФ с указанием возникающих аномалий.
10. Условия возникновения аномалий в отношении, нормализованном до 3НФ, определение НФБК и пример использования.
11. Что такое многозначные зависимости: определение и условия возникновения. Нормализация отношения до 4НФ. Теорема Фейгина.
12. Что такое зависимость соединения и в чем суть свойства n-декомпозируемости отношения? Определение 5НФ.

13. Язык SQL, его структура, преимущества и недостатки.

14. Основные операторы языка SQL и способы их использования.

15. Каким образом могут быть объединены два отношения в языке SQL (виды объединений)?

16. Какие алгоритмы соединения таблиц известны? Каковы их преимущества и недостатки?

17. Для данных значений переменных отношений применить операции реляционной алгебры:

R:

A	B
1	3
4	6
7	3

S:

B	C
1	2
3	7
4	6

18. Для переменной-отношения R(A,B,C,D,E,F) определено множество ФЗ: S = {A→BC, B→E, CD→EF } Принадлежит ли ФЗ AD→F множеству S^+?

19. Дана переменная отношения R(A,B,C,D,E). Эквиваленты ли два множества ФЗ: S_1={A→B, AB→C, D→AC, D→E} и S_2={A→BC, D→AE}?

20. Найдите неприводимое множество ФЗ отношения R(A,B,C,D,E,F), при условии, что S = {AB→C, C→A, BC→D, ACD→B, BE→C, CE→FA, CF→BD, D→EF}.

21. Дано отношение R(A, B, C, D, E, F, G, H, I, J) с множеством ФЗ S = {ABD→E, AB→G, B→F, C→J, CJ→I, G→H}. Найти потенциальные ключи.

22. Дано отношение R(A, B, C, D, E, F, G) с множеством ФЗ: S = {A→B, AE→F, BC→DE, AEF→G}. Является ли {A, C} потенциальным ключом?

23. Провести нормализацию заданного ненормализованного отношения с информацией о студентах до 5НФ с указанием первичных ключей в каждой нормальной форме, а также с примерами возникающих аномалий. Множество ФЗ для отношения: S = {{ЗК}→{ФИО, ДР}, {ЗК}→{Спец код}, {ЗК, Спец код}→{Группа}, {Группа}→{Курс}}. Предположим, что в отношении имеются много-

значная зависимость {ФИО, ДР} –›› {Тел.} | {Адреса} и зависимость соединения *{Спец код, Дисциплина, Сем.}.

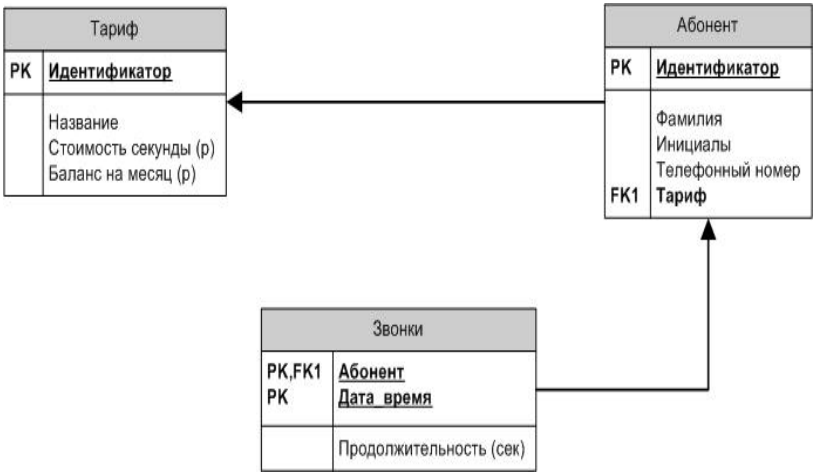
ЗК	ФИО	ДР	Спец код	Курс	Группа	Дисциплина	Сем.	Тел.	Адреса
012	Иванов И.И.	01.01.91	012345	3	АИ32	Математика	1	27-27 26-25	Соколова, 1 Пушкинская, 7
012	Иванов И.И.	01.01.91	012345	3	АИ32	Математика	2	27-27 26-25	Соколова, 1 Пушкинская, 7
012	Иванов И.И.	01.01.91	012345	3	АИ32	Физика	2	27-27 26-25	Соколова, 1 Пушкинская, 7
012	Иванов И.И.	01.01.91	012345	3	АИ32	История	1	27-27 26-25	Соколова, 1 Пушкинская, 7
023	Иванов И.И.	01.01.91	543210	1	ИБ12	Экология	1	27-27 26-25	Соколова, 1 Пушкинская, 7
023	Иванов И.И.	01.01.91	543210	1	ИБ12	Экология	2	27-27 26-25	Соколова, 1 Пушкинская, 7
023	Иванов И.И.	01.01.91	543210	1	ИБ12	Математика	2	27-27 26-25	Соколова, 1 Пушкинская, 7
013	Петров П.П.	02.02.91	012345	3	АИ32	Физика	2	12-23 31-24	Стачки, 10 Б. Садовая, 11
013	Петров П.П.	02.02.91	012345	3	АИ32	Математика	2	12-23 31-24	Стачки, 10 Б. Садовая, 11
013	Петров П.П.	02.02.91	012345	3	АИ32	Математика	1	12-23 31-24	Стачки, 10 Б. Садовая, 11
013	Петров П.П.	02.02.91	012345	3	АИ32	Англ. язык	2	12-23 31-24	Стачки, 10 Б. Садовая, 11

24. Для заданной переменной отношения из темпоральной БД провести нормализацию до 6НФ. Первичный ключ – {ID, Время}.

ID	**Время**	Фамилия	Имя	Отчество	Пол
123	[01-09-2003:21-08-2006]	Иванова	Анна	Леопольдовна	Ж
123	[21-08-2006:28-06-2008]	Петрова	Анна	Леопольдовна	Ж
124	[01-09-2003:21-05-2007]	Сидорова	Кристина	Александровна	Ж
124	[21-05-2007:28-06-2008]	Сидоров	Константин	Александрович	М

25. Для представленной ниже инфологической модели реляционной БД «Абоненты» написать SQL-запрос, выводящий количество абонентов, у которых отрицательный баланс за заданный месяц, т.е. тех абонентов, у которых превышен выделяемый им тарифом баланс. Месяц задается своими первым и последним днями. На диаграмме сущность «Абонент» представляет собой информацию об абонентах, «Тариф» - содержит информацию о тарифных планах, «Звонки» - представляет историю всех звонков абонента за все вре-

мя подключения. В каждом тарифном плане указано, какой баланс средств может быть истрачен абонентом за месяц, а также стоимость одной секунды разговора. Для каждого звонка определяется его длительность, а также стоимость одной секунды разговора. Тарификация с первой секунды.



26. Для представленной в задании 25 инфологической модели реляционной БД «Абоненты» написать SQL-запрос, выводящий количество абонентов, у которых продолжительность разговоров за заданный месяц превысило 100 минут. Месяц задается своими первым и последним днями.

3. ИСПОЛЬЗОВАНИЕ БАЗ ДАННЫХ

3.1. Области приложения баз данных и примеры их использования

Несмотря на то, что сегодня БД используются практически во всех сферах человеческой деятельности (медицина, СМИ, химия, топология, история, литература и т.д.) преимущественно рассматривают применение БД в рамках различных АИС. Поэтому в данном параграфе обзорно приведены примеры использования БД с точки зрения классификации ИС, в состав которых они входят.

По характеру обработки данных ИС принято делить на две группы:

- *информационно-поисковые ИС* (ИПС), в которых нет сложных алгоритмов обработки данных, а основным назначением таких систем является эффективный поиск и удобное представление информации для пользователей. Такие системы применяются повсеместно (электронный документооборот, электронные библиотеки, аналитика и финансы, химия и т.д.), но наиболее активно в настоящий момент развиваются поисковые системы Интернета;
- *решающие ИС* или *системы обработки данных*, которые используют в своей работе программные средства, выполняющие сложную обработку данных. Наиболее яркими представителями этой группы являются автоматизированные системы управления и системы поддержки принятия решений.

По сферам применения ИС обычно подразделяют на четыре группы:

- *информационные системы оперативной обработки транзакций* (англ. Online Transaction Processing – OLTP) – системы, поддерживающие большой поток данных и при этом обеспечивающие минимально возможное время отклика на запросы конечных пользователей. Они предназначены для обработки данных (информация, документы, операции и т.д.) в режиме реального времени, в связи с этим требования к уровню нормализации отношений в таких системах возрастают. Примерами таких систем являются автоматизированные банковские системы, системы планирования ресурсов предприятия, системы управления биржевыми операциями, а также системы заказа билетов;

- *информационные системы оперативной аналитической обработки данных* (англ. Online Analytical Processing – OLAP) – системы управления большими массивами данных, структурированных по многомерному принципу, что позволяет обеспечивать большую скорость обработки данных. Системы применяются в программных продуктах финансового планирования и в бизнес-аналитике;
- *информационно-справочные системы* (справочно-правовые системы для бизнеса, поисковые системы Интернет и т.д.);
- *офисные информационные системы* предназначены для управления делопроизводством, перевод документов в электронный вид, управление документооборотом.

3.2. Механизмы доступа к данным

Обзор механизмов доступа к данным начнем с рассмотрения архитектуры компонентов, используемых в ОС компании Microsoft (Microsoft/Windows Data Access Components – MDAC/WDAC). Схематично архитектура изображена ниже (рис.10).

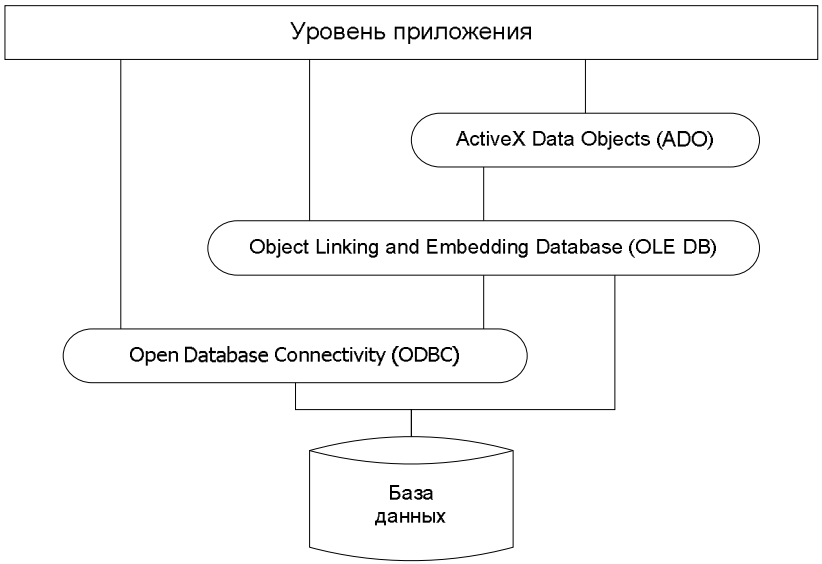


Рис.10. Архитектура доступа к данным

Наиболее низким уровнем является программный интерфейс ODBC. Он основан на *интерфейсе уровня вызовов* (англ. Call-Level Interface, CLI), который в свою очередь описывает, как именно программа должна отправлять SQL-запросы к системе управления базами данных (СУБД) и как должен быть обработан возвращаемый набор данных. Основная цель разработки ODBC – обеспечение максимального уровня совместимости разрабатываемых приложений конечных пользователей с различными источниками данных. Последнее означает, что программное средство, использующее рассматриваемый API, будет корректно работать независимо от особенностей конкретной СУБД, поддерживающей доступ к данным в БД по интерфейсу ODBC. Приложение конечного пользователя вызывает функции, имена которых определены в интерфейсе ODBC, при этом выполняется специализированный программный модуль, реализующий эти функции для конкретной СУБД, называемый *драйвером* БД. Использование драйверов позволяет оградить приложение и программиста их разрабатывающего от необходимости учитывать особенности каждой конкретной СУБД, а также исключает необходимость повторной компиляции этих приложений при изменении источника данных.

Основу архитектуры программного интерфейса ODBC составляют следующие понятия:

- *хэндл* (англ. handle) – абстрактная ссылка на ресурс памяти вычислительной машины, которая в ODBC используется для адресации среды окружения, соединения с СУБД, отдельных инструкций или дескрипторов данных;
- *буфер* (англ. buffer) – часть памяти вычислительной машины, используемой при передаче данных между драйвером БД и приложением конечного пользователя;
- *тип данных* (англ. data type) – сущность, обладающая идентификатором и определяющая множество значений, способ их представления в памяти, а также набор допустимых над этими значениями операций, при этом ODBC поддерживает типизацию данных в стиле C и стиле SQL;
- *уровень соответствия* (англ. conformance level) – абстрактное понятие, определяющее возможности функций конкретного драйвера; основной целью деления на уровни является информирование приложения о том, возможности какого уровня соответствия доступны ему в случае использования выбранного драйвера, при этом поддерживаются области соответствия ODBC-интерфейса (уро-

вень ядра, первый уровень и второй уровень) и SQL-грамматики (начальный, FIPS переходный, промежуточный или полный в соответствии со стандартом SQL-92);

- *свойство* (англ. attribute) – информационная характеристика среды выполнения, соединения или отдельных операторов:
  - свойство среды выполнения влияет на всю среду (например, открыт ли пул соединений);
  - свойство соединения локализовано для каждого конкретного соединения (например, как долго драйвер может пытаться установить соединение с СУБД перед тайм-аутом);
  - свойство оператора определено в отдельности для конкретного оператора (например, должны ли операторы быть выполнены асинхронно);
- *таблица*, *представление* (англ. table, view) – определяются в соответствии с терминологией реляционной модели данных.

Алгоритм использования ODBC в качестве механизма доступа к данным заключается в выполнении следующей последовательности шагов:

1. Установление соединения с источником данных (CONNECT).
2. Инициализация приложения (INITIALIZE).
3. Создание и выполнение операторов языка SQL (EXECUTE).
4. Получение результата запроса (FETCH RESULTS) или числа измененных записей (FETCH ROW COUNT).
5. Фиксация транзакции (COMMIT).
6. Разрыв соединения (DISCONNECT).

Рассмотрим пример использования ODBC при разработке программных средств на примере библиотеки QT (C++) и на платформе .NET (C#).

Следуя алгоритму использования ODBC, представленному выше, на первом шаге требуется выполнить подключение к источнику данных. Для этого требуется указать название драйвера, используемого при подключении:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QODBC", "MDB1");
```

Второй необязательный параметр позволяет задать имя соединения. После того, как объект, выполняющий соединение, создан в памяти, требуется указать его атрибуты. Например, в случае выполнения подключения к файлу с именем «trace.mdb», созданному в MS Access 2003, требуется указать:

```
db.setHostName("localhost");
db.setDatabaseName("DRIVER={Microsoft Access Driver (*.mdb)};FIL={MS Ac-
cess};DBQ=trace.mdb");
```

После того, как все параметры соединения определены, требуется открыть соединение вызовом соответствующего метода:

```
bool conn_res = db.open();
```

Логическая переменная conn_res принимает значение true в случае успешного соединения и false – в случае, если при соединении возникла ошибка. Для того чтобы вывести сообщение о полученной ошибке, можно использовать метод lastError() :

```
If (!conn_res) {
  qDebug() << db.lastError().text() << endl;
}
```

После того, как соединение успешно установлено, становится возможным выполнение SQL-запросов к БД. Для этого возможно использовать экземпляр класса QSqlQuery:

```
QSqlQuery query(db);
query.exec("INSERT INTO empl(id, name, salary) VALUES(33, 'Иванов', 2400)");
query.exec("SELECT id, name, salary FROM empl WHERE salary>=1000");
```

Для получения и обработки результатов выполнения запроса используется метод QSqlQuery::next(), который переводит курсор на очередную запись результирующего набора данных или возвращает false, если достигнут его конец [2]:

```
while( query.next() ) {
  qint64 id = query.value(0).toLongLong();
  QString name = query.value(1).toString();
  double salary = query.value(2).toDouble();
}
```

Метод QSqlQuery::value(номер_столбца) возвращает значение типа QVariant, соответствующее значению атрибута, имя которого расположено под указанным в качестве параметра метода номером запрошенном наборе данных. Полученное значение можно преобразовать к нужному типу данных, используя соответствующие методы (toInt(), toLongLong(), toString() и т.д.).

Число записей, которые возвращает запрос SELECT, возможно узнать с использованием метода QSqlQuery::size(). Метод size() вернет -1, если во время выполнения запроса произошла ошибка. Для запросов модификации данных (INSERT, UPDATE, DELETE) число измененных записей можно получить с использованием метода QSqlQuery::numRowsAffected().

Если СУБД поддерживает механизм транзакций, то для начала транзакции используется метод QSqlDatabase::transactioin(), для ее подтверждения – QSqlDatabase::commit(), а для отката – QSqlDatabase::rollback().

Для закрытия соединения с БД используется метод QSqlDatabase::close(). Вызовом данного метода программа сообщает СУБД, что необходимо закрыть соединение с БД и выгрузить все имеющиеся в ОЗУ буферы.

Для работы с данными с использованием API ODBC в библиотеке .NET имеется пространство имен System.Data.Odbc, которое подключается командой:

```
using System.Data.Odbc;
```

Для доступа к данным требуется создать экземпляр класса OdbcConnection и инициализировать путь до источника данных:

```
OdbcConnection DbConnection = new OdbcConnection("DRIVER={Microsoft Access
Driver (*.mdb)};FIL={MS Access};DBQ=trace.mdb);
```

После создания объекта, обеспечивающего соединение с базой данных, требуется вызвать метод, открывающий соединение:

```
DbConnection.Open();
```

Теперь возможно определить необходимую к исполнению команду. Для этого следует создать экземпляр класса OdbcCommand, а также в случае, если требуется обработать полученный от СУБД ответ (запрос SELECT) – создать экземпляр класса OdbcDataReader и произвести необходимую обработку:

```
OdbcCommand DbCommand = DbConnection.CreateCommand();
DbCommand.CommandText = "INSERT INTO empl(id, name, salary) VALUES(33,
'Иванов', 2400)";
DbCommand.ExecuteReader();
DbCommand.CommandText = "SELECT id, name, salary FROM empl WHERE sal-
ary>=1000";
using(OdbcDataReader reader = DbCommand. ExecuteReader()) {
 // вывод на экран названий полей, включенных в выборку
 int fCount = reader.FieldCount;
 Console.Write(":");
 for ( int i = 0; i < fCount; i ++ )
 {
  String fName = reader.GetName(i);
  Console.Write( fName + ":");
 }
 Console.WriteLine();
 // вывод на экран результатов выполнения запроса
 while( reader.Read() )
 {
```

```
        Console.Write( ":" );
        for (int i = 0; i < fCount; i++)
        {
          String col = reader.GetString(i);
          Console.Write(col + ":");
        }
        Console.WriteLine();
      }
    }
```

По окончании работы с БД следует закрыть соединение с использованием метода Close.

Следующим уровнем в иерархии технологий доступа к данным компании Microsoft является низкоуровневый интерфейс OLE DB. Он может быть использован для доступа к различным источникам данных (реляционным и нереляционным, текстовым и графическим данным, файловым системам и бизнес-объектам). Данный набор программных интерфейсов основан на использовании объектной модели компонентов (COM), которые инкапсулируют в себе различные сервисы управления данными и при этом предоставляют однотипный доступ к различным источникам данных.

Объектная модель OLE DB состоит из:

- *источника данных* (Data Source) – объекта, инкапсулирующего информацию, связанную с соединением и обеспечивающего предоставление данных из источника данных;
- *контекста* или *сессии* (Session) – объекта, выполняющего роль фабрики классов (обеспечивающей создание, управление и удаление объектов) для создания *команд* и *наборов данных*, при этом для одного источника данных может быть открыто несколько сессий;
- *команд* (Command) – объектов, предназначенных для выполнения текстовых команд (преимущественно в формате SQL-запросов), при этом в рамках одной сессии может быть выполнено множество команд;
- *набора данных* (Rowset) – объект, обеспечивающий представление данных из источников данных, полученных с использованием некоторых команд в рамках определенного контекста в форме плоской таблицы (набор строк, каждая из которых содержит одну или несколько колонок), а также предоставляющий функции по добавлению, модификации и удалению данных в виде строк таблицы.

Рассмотренные технологии ODBC и OLE DB считаются хорошими интерфейсами передачи данных, но их использование в каче-

стве программных интерфейсов сопряжено с рядом ограничений, так как они являются низкоуровневыми. ADO – это высокоуровневая объектная модель, которая создает дополнительный уровень абстракции между приложениями конечных пользователей и источниками данных. В платформе .NET имеется своя версия данной модели – ADO.NET, которая считается наиболее продвинутой [16].

Алгоритм использования ADO.NET состоит из установления соединения с хранилищем данных, создания и заполнения данными объекта DataSet, отключения от хранилища данных с фиксацией изменений, произведенных в объекте DataSet, в источнике данных.

Объект DataSet представляет локальный набор таблиц и информации об отношениях между ними, что позволяет удобно создать в клиентском приложении полную копию удаленной базы данных. Объект состоит из коллекции таблиц (свойство класса Tables), коллекции связей между таблицами (свойство класса Relations) и коллекции свойств (свойство класса ExtendedProperty). Таблицы предназначены для взаимодействия с данными в БД, связи в реляционной модели данных представляются ограничением «внешний ключ», а свойства обеспечивают возможность ассоциирования любой дополнительной информации с экземпляром DataSet. Рассмотрим листинг доступа к источнику данных из предыдущего примера с использованием ADO.NET и провайдера OLE DB.

```
using System;
using System.Data;
using System.Data.OleDb;
public class MainClass {
  public static void Main()
  {
    // создание объекта-соединения с источником данных
    OleDbConnection myConn = new OleDbConnection ("Provider=
Microsoft.Jet.OLEDB.4.0;Data Source=trace.mdb");
    // в рамках созданного контекста определяем объект для выполнения команды
    OleDbCommand myComm = new OleDbCommand("SELECT id, name, salary
FROM empl WHERE salary>=1000", myConn);
    // определяем объект-адаптер, используемый для выполнения команды и
    // заполнения источников данных
    OleDbDataAdapter myAdap = new OleDbDataAdapter(myComm);
    // выполнение соединения
    myConn.Open();
    // создать и заполнить набор данных
    DataSet myDS = new DataSet();
```

```
    myAdap.Fill(myDS, "Empl");
    // обработка данных
    DataRowCollection drc = myDS.Tables["Empl"].Rows;
    foreach(DataRow dr in drc)
    {
      Console.WriteLine(dr[0] + " " + dr[1]);
    }
   myConn.Close();
  }
 }
```

В заключение необходимо сказать, что помимо представленных механизмов для доступа к данным существуют:

- DAO (Data Access Objects) – еще одна технология доступа к данным компании Microsoft, которая является менее универсальной чем OLE DB\ADO, однако, часто используется при разработке приложений для взаимодействия с файл-серверными БД (например, MS Access), при этом поддержка клиент-серверных источников данных в данной технологии не достаточно эффективна;
- JDBC (Java Data Base Connectivity) – программный кросс-платформенный мобильный интерфейс для доступа к БД с использованием платформы Java;
- BDE (Borland Database Engine) – технология доступа к данным фирмы Borland, реализованная в виде динамически подключаемых библиотек, своего рода, аналог ODBC при разработке приложений БД в среде Delphi.

3.3. Использование БД в web-приложениях

Большинство web-проектов сегодня основаны на использовании СУБД в качестве системы, обеспечивающей управление и оперативный доступ к данным, хранимым в БД и сопровождаемым web-приложением. Работа с различными источниками данных обеспечивается специализированными драйверами, которые предоставляют собственный программный интерфейс для взаимодействия с СУБД.

Принцип взаимодействия с СУБД при разработке web-приложений заключается в *установлении соединения* с БД, *выполнения определенных команд* и *закрытии соединения*. Ниже представлены функции, которые могут быть использованы для взаимодействия с БД, управляемой СУБД MySQL, на языке PHP 5.1 [**17**, **22**]:

- mysql_connect ([string $hostname] [, string $user] [, sting $password]) – установление соединения с сервером БД, расположен-

ным на хосте $hostname под именем пользователя $user с паролем $password, при этом идентификатор соединения является возвращаемым значением;

- mysql_close(int $connection_id) – закрытие соединения для переданного идентификатора;
- mysql_select_db(string $db [, int $id]) – выбор БД на сервере, с которой будет происходить работа PHP-скрипта, при этом второй параметр используется при одновременном открытии нескольких соединений;
- mysql_query(string $query) – выполнить запрос к БД, при этом результат выполнения запроса будет представлен в качестве возвращаемого значения функции и может быть обработан специальными функциями:
  - mysql_result(resource $result, int $row [, mixed $field]) – универсальная функция для получения необходимого элемента из набора записей;
  - mysql_fetch_array(resource $result) – получить очередную запись из результатов запроса в виде пронумерованного массива, в котором числовые значения ключей соответствуют порядку выводимых полей, которым (в свою очередь) соответствуют значения, составляющие запись;
  - mysql_fetch_assoc (resource $result) – получить очередную запись из результатов запроса в виде ассоциативного массива, в котором ключи соответствуют именам выводимых полей, которым (в свою очередь) соответствуют значения, составляющие запись;
- mysql_num_rows(resource $result) – возвращает количество полученных в результате выполнения запроса SELECT записей;
- mysql_affected_rows() – возвращает количество затронутых выполнением последнего запроса на модификацию данных (INSERT, DELETE, UPDATE).

Рассмотрим применение этих функций на следующем примере.

```
<?php
$host = "localhost";
$user = "user";
$password = "secret_password";
// Производим попытку подключения к серверу MySQL:
if (!mysql_connect($host, $user, $password))
{
  echo "<h2>MySQL Error!</h2>";
```

```
  exit;
}
// Выбираем базу данных:
mysql_select_db($db);
// Выводим заголовок таблицы:
echo "<table border=\"1\" width=\"100%\" bgcolor=\"#FFFFE1\">";
echo "<tr><td>Email</td><td>Имя</td><td>Месяц</td>";
echo "<td>Число</td><td>Пол</td></tr>";
// SQL-запрос:
$q = mysql_query ("SELECT * FROM mytable");
// Выводим таблицу:
for ($c=0; $c<mysql_num_rows($q); $c++)
{
echo "<tr>";
$f = mysql_fetch_array($q);
echo "<td>$f[email]</td><td>$f[name]</td><td>$f[month]</td>";
echo "<td>$f[day]</td><td>$[s]</td>";
echo "</tr>";
}
echo "</table>";
?>
```

Библиографический список

1. ACM, IBM Dictionary of Computing, 10th edition. – 1993.

2. Codd, E.F. Data Models in Database Management. Proc. Workshop in Data Abstraction, Databases, and Conceptual Modelling: ACM SIGART Newsletter No. 74, 1981.

3. Codd, E.F. Is Your DBMS Really Relational? *ComputerWorld*, 14. – 1985.

4. Dev Center – Desktop. [Электронный ресурс] / Microsoft ODBC: Developing Applications: Электрон. дан. – 2012. – Режим доступа: http://msdn.microsoft.com/en-us/library/windows/desktop/ms709378(v=vs.85).aspx. – Загл. с экрана. – Яз. англ.

5. Graph databases [Электронный ресурс] / Wikipedia. – Электрон. дан. – 2013. – Режим доступа:
http://en.wikipedia.org/wiki/Graph_databases. – Загл. с экрана. – Яз. англ.

6. Worsley C. John and Drake D. Joshua. PostgreSQL для профессионалов. – СПб: Питер, 2003. – 498 с.

7. ГОСТ Р ИСО МЭК ТО 10032-2007: Эталонная модель управления данными.

8. Дейт К., Дж. Введение в системы баз данных, 8-е изд. / пер. с англ. - М.: Вильямс, 2006. - 1072 с.

9. Земсков Ю. Qt 4 на примерах. – СПб.: БХВ-Петербург, 2008. – 608 с.

10. Жуков А.И. Использование информационных систем и технологий в целях удовлетворения информационных потребностей / А.И. Жуков, А.Г. Сорокин. – Красноярск: Научно-инновационный центр, 2012. – С. 5-39.

11. Модель данных [Электронный ресурс] / Википедия. – Электрон. дан. – 2013. – Режим доступа:
http://ru.wikipedia.org/wiki/Модель_данных. – Загл. с экрана. – Яз. рус.

12. Нормальная форма [Электронный ресурс] / Википедия. – Электрон. дан. – 2013. – Режим доступа:
http://ru.wikipedia.org/wiki/Нормальная_форма. – Загл. с экрана. – Яз. рус.

13. Подвальный С.Л., Сергеева Т.И., Гранков М.В. Базы данных: модели данных, SQL, проектирование: учеб. пособие. – Ростов н/Д: Издательский центр ДГТУ, 2007. – 320 с.

14. Проектирование баз данных [Электронный ресурс] / Википедия. – Электрон. дан. – 2013. – Режим доступа:

http://ru.wikipedia.org/wiki/Проектирование_баз_данных. – Загл. с экрана. – Яз. рус.

15. Таненбаум Э. Современные операционные системы. – 2-е изд. – СПб.: Питер, 2002. – 1040 с.

16. Троелсен Э. C# и платформа .NET. Библиотека программиста – СПб.: Питер, 2006. – 796 с.

17. Функции PHP для работы с MySQL / PHP.SU: Электрон. дан. – 2012. – Режим доступа: http://www.php.su/mysql/?functions. Загл. с экрана. – Яз. рус.

18. Хансен Гэри, Хансен Джэймс. Базы данных: разработка и управление / пер. с англ. - М.: БИНОМ, 1999. - 704 с.

19. Харитонова И.А., Михеева В.Д. Microsoft Access 2000. – СПб.: БХВ, 2002. – 1088 с.

20. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: учебник для высших учеб. заведений. – СПб.: КОРОНА принт, 2000. – 416 с.

21. Чардин П. Многоверсионность данных и управление параллельными транзакциями [Электронный ресурс] / CITForum: Электрон. дан. – 2005. – Режим доступа:
http://citforum.ru/database/articles/multiversion/. – Загл. с экрана. –Яз. рус.

22. Шапошников И.В. PHP 5.1. Учебный курс. – СПб.: Питер, 2007. – 192 с.

23. Шестая нормальная форма [Электронный ресурс] / Википедия. – Электрон. дан. – 2013. – Режим доступа:
http://ru.wikipedia.org/wiki/Шестая_нормальная_форма. – Загл. с экрана. – Яз. рус.

Оглавление

Предисловие 3

1. ВВЕДЕНИЕ В ТЕОРИЮ БАЗ ДАННЫХ 4
   1.1. Основные понятия и определения 4
      1.1.1. Информационные системы и базы данных 4
      1.1.2. Уровни абстракции в базах данных 6
      1.1.3. СИСТЕМЫ УПРАВЛЕНИЕ БАЗАМИ ДАННЫХ 8
      1.1.4. Компоненты системы баз данных 11
   1.2. Модели данных 14
   1.3. Основные этапы проектирования БД 17
      1.3.1. Концептуальное проектирование 17
      1.3.2. Логическое проектирование 21
      1.3.3. Физическое проектирование 22
   1.4. Использование базы данных в многопользовательском режиме 22
      1.4.1. Транзакции 22
      1.4.2. Три проблемы параллельности 24
      1.4.3. Блокировки 26
      1.4.4. Многоверсионность данных 27
      1.4.5. Распределенная обработка данных 30
   1.5. Вопросы для самоконтроля 34

2. РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ: ПРОЕКТИРОВАНИЕ, СОЗДАНИЕ, ИСПОЛЬЗОВАНИЕ 36
   2.1. Реляционная модель данных 36
      2.1.1. Основные определения 36
      2.1.2. Виды реляционных отношений 38
      2.1.3. Реляционная алгебра 39
      2.1.4. Правила Кодда 42
   2.2. Формальный подход к проектированию реляционных баз данных 44
      2.2.1 Функциональные зависимости 45
      2.2.2. Правила Армстронга и неприводимое множество ФЗ 46
      2.2.3. Первичный ключ 47
      2.2.4. Внешний ключ 50
      2.2.5. Механизм нормализации и нормальные формы . 51

2.3. Основы использования языка SQL 65
2.3.1. Структура языка SQL 65
2.3.2. Оператор подъязыка запросов 67
2.3.3. Операторы подъязыков определения и модификации данных 75
2.3.4. Преимущества и недостатки языка SQL 79
2.4. Вопросы и задания для самоконтроля 80

3. ИСПОЛЬЗОВАНИЕ БАЗ ДАННЫХ 84
3.1. Области приложения без данных и примеры их использования 84
3.2. Механизмы доступа к данным 85
3.3. Использование БД в web-приложениях 92

Библиографический список 96

Учебное издание

Гранков Михаил Васильевич,
Жуков Александр Игорьевич

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ СУБД ПРИ РАЗРАБОТКЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Редактор В.Ф. Лавриченко
Компьютерная обработка: И.В. Кикичева
Тем. план 2013 г.

В печать 03.09.2013.
Объём 6,2 усл. п.л. Офсет. Формат 60x84/16.
Бумага тип №3. Заказ №817. Тираж 100 экз. Цена свободная

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина,1.