Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

Е.М. ДАВЫДОВА, Н.А. НОВГОРОДОВА

БАЗЫ ДАННЫХ

УЧЕБНОЕ ПОСОБИЕ

2008

ДАВЫДОВА Е.М., НОВГОРОДОВА Н.А.
Базы данных: Учебное пособие. — Томск: Томск. гос. ун-та систем управления и радиоэлектроники, 2008. — 127 с.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ В БАЗЫ ДАННЫХ

Информатика — это наука о применении компьютеров в различных сферах деятельности, то есть наука о компьютерных (иногда говорят — новых информационных) технологиях.

Информатика занимается схематичным, «формализованным» представлением информации, ее обработкой, предписаниями по ее переработке и машинами, обрабатывающими информацию. Цель информатизации состоит в разработке способов решения задач информационной обработки на вычислительных машинах, а также в разработке, организации и эксплуатации вычислительных систем.

Формирование моделей информатики нацелено на представление определенных структур, взаимодействий и процессов в какой-либо области применения (предметной области) с помощью формальных средств, таких как структуры данных, языки программирования или логические формулы. Задача информатики — исследовать свойства формальных моделей, развивать их далее и устанавливать связи между формальными моделями и реальным миром в данной предметной области в смысле постановки задачи.

1.1 Информационные системы, их разновидности

В начальный период развития ЭВМ каждый программист, разрабатывая программы, разрабатывал и свою форму представления данных, т.е. информационную модель объекта. Это приводило к тому, что каждая программа работала со своими данными что, в конечном счете, вело к избыточности и множеству информационных моделей объекта.

Важной предпосылкой создания единой информационной модели стала возможность сохранения больших объемов данных. Появились магнитные диски и удобный доступ к данным.

Введем понятия: информация и данные.

Под ***информацией*** (в бытовом понимании) понимают любые сведения о каком-либо событии, сущности, процессе и т.д., являющемся объектом операций: восприятия, передачи, преобразования или использования. Информативные образцы могут соз-

даваться в самых разнообразных формах: в форме звуковых, световых или радиоволн, электрического тока или напряжения, магнитных полей, знаков на бумажном носителе и т.д.

Поскольку существует потребность в информации при принятии решений в любой сфере деятельности: в производстве, в человеческих отношениях, в управлении и т.д., человек создал естественные информационные системы: лекции, общение между людьми, радио, газета и т.д.

В нашем сложном мире любая взаимосвязь и коррекция действий возможны только благодаря информации. Недаром говорят: «Кто владеет информацией, тот владеет всем», или: «Самое ценное, что у нас есть, — это информация».

Данные можно определить как информацию, фиксированную в определенной форме, пригодной для последующей обработки, хранения и передачи.

Совокупность различных средств, предназначенных для сбора, подготовки, хранения, обработки и выдачи информации в интересах различных пользователей и приложений, представляет собой ***информационную систему*** (ИС).

Информационная система объединяет следующие составляющие:

– языковые средства и правила, используемые для отбора и подготовки информации к вводу в ЭВМ, для отображения картины реального мира в модель данных, для работы пользователя с системой, для предоставления пользователю выдаваемой системой информации;

– информационный фонд системы;

– способы и методы организации информационных массивов и всех процессов обработки информации в системе;

– алгоритмы функционирования системы, т.е. алгоритмы всех процедур по созданию, ведению и обработке информационных массивов, алгоритмы формирования ответов на запросы, и др.;

– программное обеспечение системы, в состав которого входят программы, реализующие все алгоритмы функционирования системы;

– комплекс технических средств, функционирующих в системе;

– персонал, обслуживающий ИС.

При разработке информационных систем возникают два аспекта: инфологический и датологический. На этапе инфологического проектирования ИС решаются вопросы, о каких явлениях или объектах реального мира требуется накапливать информацию, какие их характеристики и взаимосвязи будут учитываться. Задача инфологического этапа проектирования базы данных — получение семантических (смысловых) моделей, отражающих конкретную предметную область. Таким образом, на этапе инфологического проектирования решается вопрос о предметной области. Датологический аспект решает вопросы о представлении в памяти машины информационной системы, т.е., исходя из существующих средств, решаются вопросы хранения и обработки информации, модели представления информации, методы ее преобразования и формируются правила смысловой интерпретации.

Поскольку области применения информационных систем разнообразны, разнообразны и их свойства и особенности, присущие каждой ИС. Среди множества факторов, определяющих совокупность свойств конкретной ИС, можно выделить три основных: технический уровень системы; характер обрабатываемой информации; целевые функции, т.е. круг задач, для решения которых данная система предназначена.

Рассмотрим последовательно все три фактора.

По техническому уровню ИС разделяются на ручные, механизированные, автоматизированные и автоматические.

В ручных ИС все процессы обработки информации осуществляются вручную. Эти ИС имеют небольшой объем и хранятся на специальных карточках (например, перфокарты, личные карточки студентов, картотеки) различных типов. Примером ручной ИС может служить библиотечный каталог.

В механизированных ИС для обработки и поиска информации используются различные средства механизации, а именно счетно-перфорационные машины. Кроме того, в комплект технических средств входит набор перфорационных машин, посредством которых информация заносится на перфокарты.

В автоматизированных и автоматических ИС для хранения, обработки и поиска информации используются ЭВМ. Кроме этих

операций, ЭВМ используется при выполнении операций, связанных со сбором, подготовкой и передачей информации, а также выдачей ее пользователю. При функционировании автоматизированных информационных систем (АИС), являющихся наиболее распространёнными, на различных этапах технологического процесса обработки информации принимает участие человек, который является полноправным партнером ИС. В автоматических ИС все процессы протекают без участия человека. Обычно автоматические ИС используются в составе крупных АСУ технологическими процессами и объектами. «Партнерами» автоматических ИС являются роботы, станки с числовым программным обеспечением, технологические процессы. Входная информация в таких системах представляется в форме сигналов или каких-либо физических величин, выходная информация используется для управления или регулирования, например, технологического процесса.

По характеру обрабатываемой информации ИС делятся на документальные и фактографические.

Документальные ИС служат для работы с документами на естественном языке: монографиями, научными отчетами, диссертациями, текстами законодательных актов и т.д. Основной задачей документальных ИС является поиск документов по их содержанию. Это требует понимания системой смысла запросов.

Документальные модели данных соответствуют представлению о слабо структурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке[7].

Модели, основанные на языках разметки документов, связаны прежде всего со стандартным общим языком разметки — SGML (Standard Generalised Markup Language), который был утвержден ISO в качестве стандарта еще в 80-х годах. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тегов (ссылок), их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования тегов осуществляется при помощи специального набора правил, называемых DTD-описаниями, которые используются программой клиента при разборе документа. Для каждого класса документов определяется свой набор правил, описывающих грамматику соответствующего языка разметки. С помощью SGML мож-

но описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Но ввиду своей сложности SGML использовался в основном для описания синтаксиса других языков (наиболее известным из которых является HTML), и немногие приложения работали с SGML-документами напрямую.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа и имеет некий ограниченный набор инструкций — тегов, при помощи которых осуществляется процесс разметки. Инструкции HTML в первую очередь предназначены для управления процессом вывода содержимого документа на экране программы-клиента и определяют тем самым способ представления документа, но не его структуру. В качестве элемента гипертекстовой базы данных, описываемой HTML, используется текстовый файл, который может легко передаваться по сети с использованием протокола HTTP. Эта особенность, а также то, что HTML является открытым стандартом и огромное количество пользователей имеет возможность применять возможности этого языка для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его сегодня главным механизмом представления информации в Интернете.

XML (Extensible Markup Language) — это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Он используется в качестве средства описания грамматики других языков и контроля за правильностью составления документов. То есть сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания.

Тезаурусные модели основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

В процессе индексирования документов используется и составляется ***тезаурус*** — словарь, содержащий перечисление ос-

новных лексических единиц предметной области и описание парадигматических отношений между ними. ***Парадигматические отношения*** — это семантические отношения, существующие между лексическими единицами языка независимо от контекста. Поиск нужного документа производится по соответствующему запросу.

Дескрипторные модели — самые простые из документальных моделей, они широко применялись на ранних стадиях использования документальных баз данных. В этих моделях каждому документу соответствовал дескриптор — описатель. ***Дескриптор*** — это некоторое множество слов или словосочетаний, которые соответствуют основным понятиям предметной области и включены в ее тезаурус**.**

Дескриптор имел жесткую структуру и описывал документ в соответствии с теми характеристиками, которые требуются для работы с документами в разрабатываемой документальной БД.

Как определить о чем идет речь, когда употребляется слово *РАК* — животное, заболевание или созвездие? Для распознавания того, о чем идет речь, необходимо составить тезаурус для каждого из трех вариантов поиска.

При поиске нужного документа возможны ситуации: неточность поиска (информационный шум); неполнота (не все документы найдены); релевантные документы (соответствующие запросу).

Качество поиска документа оценивается по точности поиска и полноте. Полнота поиска определяется по формуле:

$$R=A/C,$$

где A — число документов, релевантных запросу, C — общее число документов, имеющихся в системе.

А точность поиска нужного документа определяется по формуле:

$$P=A/L,$$

где L — общее количество документов, выданных в ответ на запрос.

В фактографических ИС хранимая и обрабатываемая информация представляет собой конкретные сведения, факты, примеры, результаты измерений, справочные и статистические данные. Информация часто носит оперативный характер. Она выда-

ется пользователю или прикладным программам для обработки. Фактографические ИС реализуются банками данных.

Банк данных — это автоматизированная информационная система, включающая в свой состав комплекс специальных методов и средств (математических, информационных, программных, языковых, организационных и технических) для поддержания динамической модели предметной области с целью обеспечения информационных запросов пользователей[2].

В результате развития технологии ИС система доступа к данным и данные, которые накапливаются в системе, были логически разделены. Поэтому БнД состоит из двух частей:

1) базы данных (БД), которая является датологическим представлением информационной модели предметной области;

2) системы управления базой данных (СУБД), с помощью которой и реализуется централизованное управление данными, хранимыми в базе, доступ к ним и поддержание их в состоянии, соответствующем состоянию предметной области.

Примером фактографических систем служат: учет складских запасов, обработка заказов и товарно-транспортных накладных, учет поставок, подготовка форм статистической отчетности и т.д.

СУБД — программная система, предназначенная для создания БД, поддержания ее в актуальном состоянии, обеспечения эффективного доступа к ней. Доступ к БД возможен только посредством СУБД. СУБД общего назначения не ориентированы на определенную предметную область и предоставляются пользователю как коммерческое изделие, обладающее средствами настройки для работы с конкретной БД в условиях конкретного применения. СУБД общего назначения входит чаще всего в состав инструментального средства для создания автоматизированных информационных систем и позволяет сократить сроки разработки таких систем.

По целевым функциям (назначение самой ИС) системы подразделяются:

- информационно-справочные;
- управленческие;
- информационно-расчетные;
- информационно-логические.

Информационно-справочные системы обеспечивают информационные запросы пользователя, примером такой системы может служить система резервирования мест на авиационном и железнодорожном транспорте.

Управленческие ИС служат для решения задач АСУ предприятия: ИС больниц, автоматизированных складов, материально-технического снабжения, учета кадров, бухучета и т.д. Они, как правило, ориентированы на выдачу различного рода справок, отчетных форм для руководства предприятием.

В информационно-расчетных ИС хранимая информация используется для решения расчетных задач. Например, статистический учет и анализ, прогноз месторождений и погоды. К этим же системам относятся системы, функционирующие в составе САПР.

Информационно-логические системы способны выдавать информацию, не введенную ранее в систему в непосредственном виде, а вырабатываемую на основании логического анализа, обобщения и т.д. Это, как правило, интеллектуальные ИС.

1.2 Состав АИС

Любая АИС функционирует в окружении внешней среды, которая является источником входной и потребителем выходной информации. Начиная с входа в систему и заканчивая выходом, информационный поток проходит несколько этапов обработки:

– сбор, регистрация и первичная обработка;
– передача по каналу связи от источника к ЭВМ;
– перенос на машинные носители;
– создание и поддержка информационных фондов;
– внутримашинная обработка и формирование выходных форм;
– передача по каналу связи от ЭВМ к пользователю;
– преобразование к виду, пригодному для восприятия пользователем.

Подсистема *СБОРА И ПЕРВИЧНОЙ ОБРАБОТКИ*, как привило, занимается сбором информации на естественном языке. В результате первичной проверки отбирается информация, которой нет в АИС и которая будет в дальнейшем введена в ЭВМ. Эта

информация на первоначальном этапе обработки приводится к виду, пригодному для ввода в ЭВМ, и фиксируется на различного рода носителях.

Подсистема *ВВОДА* получает информацию от подсистемы *СБОРА* и *ПЕРВИЧНОЙ ОБРАБОТКИ* в пригодном для ввода в ЭВМ виде. Осуществляет перенос информации в систему (на машинные носители), при этом производятся контроль за правильностью переноса информации и устранение возникших ошибок.

Введенная информация размещается в ЭВМ и образует информационный фонд АИС. Над элементами информационного фонда осуществляются различные операции внутримашинной обработки: сортировка, поиск, различные арифметические и логические операции, т.е. операции, которые обеспечивают поддержание АИС в актуальном состоянии. Здесь же формируется выходная информация в соответствии с заданием на обработку. Запрос пользователя перед вводом в ЭВМ также проходит определенную обработку и рассматривается ЭВМ как задание на внутримашинную обработку.

Формирование и поддержка информационных массивов, а также все операции внутримашинной обработки осуществляются под управлением специального комплекса программ, входящих в подсистему *ВНУТРИМАШИННОЙ ОБРАБОТКИ*.

Таким образом, в систему *ОБРАБОТКИ И ХРАНЕНИЯ* информации входят: информационный фонд, подсистема внутримашинной обработки. Кроме того, сюда следует включить технические средства ЭВМ: средства ввода/вывода, средства хранения информации.

Подсистема *ВЫДАЧИ И ОТОБРАЖЕНИЯ* (вывода) обеспечивает выдачу ответа на запрос, представляя его в форме, удобной для восприятия пользователем. В состав входит комплекс программ, обеспечивающих нужный вид выходной информации, а также технические средства, на которых отображается выходная информация: принтеры, дисплеи, графопостроители, табло, индикаторы.

Подсистема *СВЯЗИ* необходима в том случае, когда источники информации, ЭВМ и пользователи удалены друг от друга. В систему связи входят каналы передачи данных: телеграфные, телефонные и компьютерные сети; удаленные терминалы: дисплеи,

специальные терминалы, абонентские пункты; программное обеспечение, позволяющее связать терминалы с центральной ЭВМ.

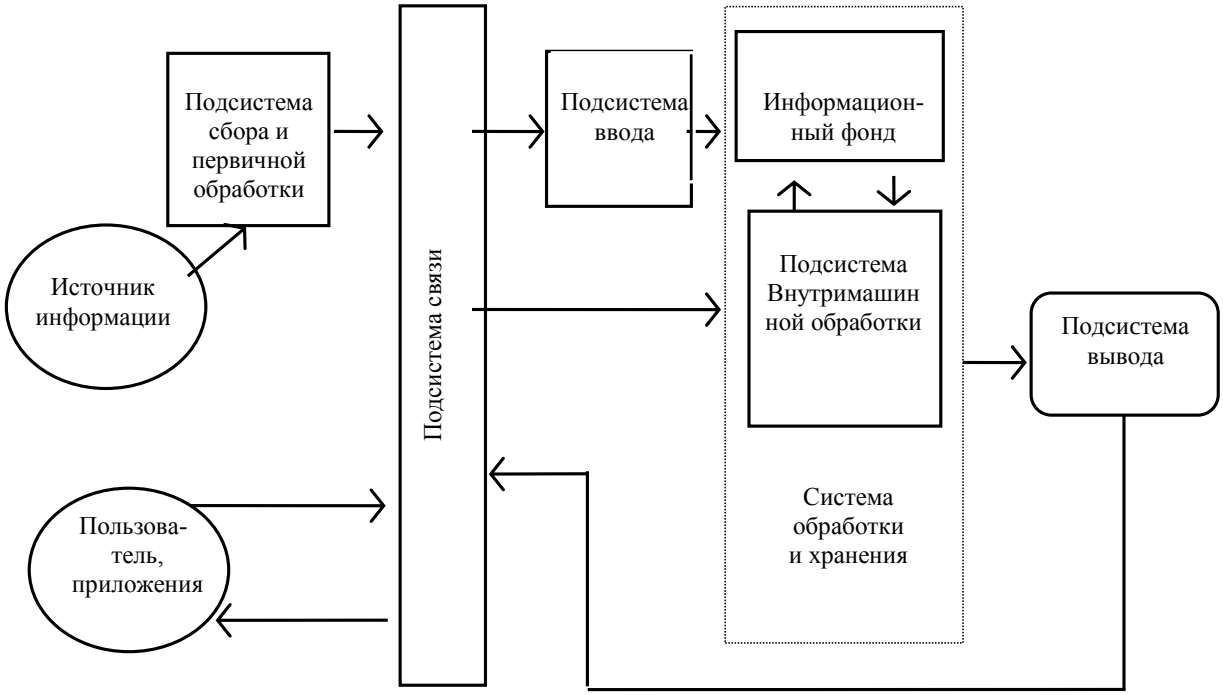


Рис. 1 — Состав АИС

1.3 Архитектура, предметная область

Реальный мир многообразен. Нет такой ЭВМ, в которую можно было бы занести весь окружающий нас мир. В своей деятельности каждый использует часть от всего его многообразия, т.е. модель, адекватно представляющую интересующую нас область. Часть реального мира, сведения о которой накапливаются в базе данных и подвергаются обработке, называют ***предметной областью.***

Определение предметной области (ПО) является начальным этапом при проектировании любой ИС. Именно на этом этапе выявляются все потребности и все пользователи, определяющие содержание будущей системы. Предметная область БД определена, если известны существующие в ней объекты, их свойства и отношения. Объектом считают любые предметы, события, понятия и т.п., которые представляют интерес для пользователей.

Проектирование БД начинается с предварительной структуризации ПО. В хорошо спроектированной схеме структура базы данных оказывается легко доступной для анализа и при модификации базы не возникает противоречий. Объекты (сущности) реального мира, о которых будет накапливаться информация, подвергаются классификации, и фиксируется совокупность их свойств (атрибутов), посредством которых будут описаны в БД типы объектов. Например, объект ЭВМ можно характеризовать такими атрибутами: тип и число процессоров, скорость вычислений, объем оперативной памяти, система команд, числом портов ввода/вывода и т.д. Кроме того, фиксируются виды отношений (взаимосвязей) между объектами.

Объектная система имеет следующие составляющие:

Объект	(Сущность)
Свойство	(Атрибут)
Объектное отношение	(Связь)

Время

Выбор объектов осуществляется в соответствии с назначением АИС. Объекты могут быть атомарные и составные. Для составного объекта должны быть определены его внутренние составляющие (которые в свою очередь могут быть объектами). Таким образом, при проектировании необходимо определить внутреннюю структуру объекта.

Каждый объект в каждый момент времени характеризуется определенным состоянием. Состояние объекта описывается с помощью некоторого набора свойств (атрибутов) и связей (отношений) с другими объектами. Каждый объект в системе отличается от других своим набором свойств. Выбор существенного атрибута или атрибутов носит субъективный характер.

Объекты имеют определенное состояние, как в отдельные моменты времени, так и в течение некоторых интервалов времени. Учитывая фактор времени при проектировании, можно в дальнейшем строить динамические модели, в которых отражается зависимость от времени составляющих объектной системы.

Объекты группируются в группы однородных объектов по структуре и поведению. Группа называется типом объектов. При

этом все экземпляры данного типа обладают одинаковыми свойствами (атрибутами). Свойства типа «наследуются» каждым экземпляром объекта данного типа. Для того чтобы отличить один объект от других объектов данного типа, необходим уникальный идентификатор, называемый первичным ключом. В качестве ключа можно использовать атрибут или группу атрибутов.

Между объектами могут существовать связи, имеющие различный содержательный смысл.

В качестве объектов в моделях ПО могут рассматриваться, как материальные объекты реальной действительности (предприятия, изделия, сотрудники и т.д.), так и нематериальные (описание явлений, рефераты статей и т.д.).

Для примера рассмотрим магазин и в качестве объекта возьмем, например, книги. Атрибутами этого товара будут наименование товара (автор и название), тематика, тираж, цена. Объектом также может быть издательство, которое выпустило данную книгу. В качестве атрибутов в этом случае будут наименование издательства, адрес, телефон, ответственное лицо. Между товаром в книжном магазине и издательством существует связь — поставка товара.

Связи могут быть обязательными и факультативными. Обязательные связи определены необходимостью связи объектов.

Совокупность типов объектов ПО и типов связей между ними характеризует типовую структуру предметной области.

При моделировании ПО проектировщик разбивает ее на ряд локальных областей, моделируя каждую область отдельно, а затем выполняет их объединение посредством связей.

Особенность современных технологий баз данных состоит в том, что в настоящее время используется трехуровневая архитектура ANSI/SPARC представления БД:

1) концептуальный уровень;

2) внешний уровень;

3) внутренний уровень.

На первом, концептуальном уровне, описываются элементы данных и связи между ними, которые являются отражением объектов и связей реального мира. При построении концептуальной модели проводится анализ информационных потребностей пользователей и определение нужных им элементов данных. На осно-

ве концептуального проектирования строится единое логическое описание всех элементов данных и отношений между ними.

Каждая группа пользователей в зависимости от потребностей получает свое представление в данных в базе данных. Каждая из групп «видит» только необходимую ей часть данных. Эта часть (внешний уровень) определяется (выводится) из полной концептуальной модели БД.

Внутренний уровень — уровень хранения БД, обеспечивает представление данных в памяти ЭВМ.

Конечный пользователь всегда работает с внешним представлением БД. За логическую и физические структуры отвечает администратор БД.

Проектирование БД начинается с концептуального уровня, когда пользовательские представления данных внешнего уровня интегрируются в единую концептуальную схему баз данных.

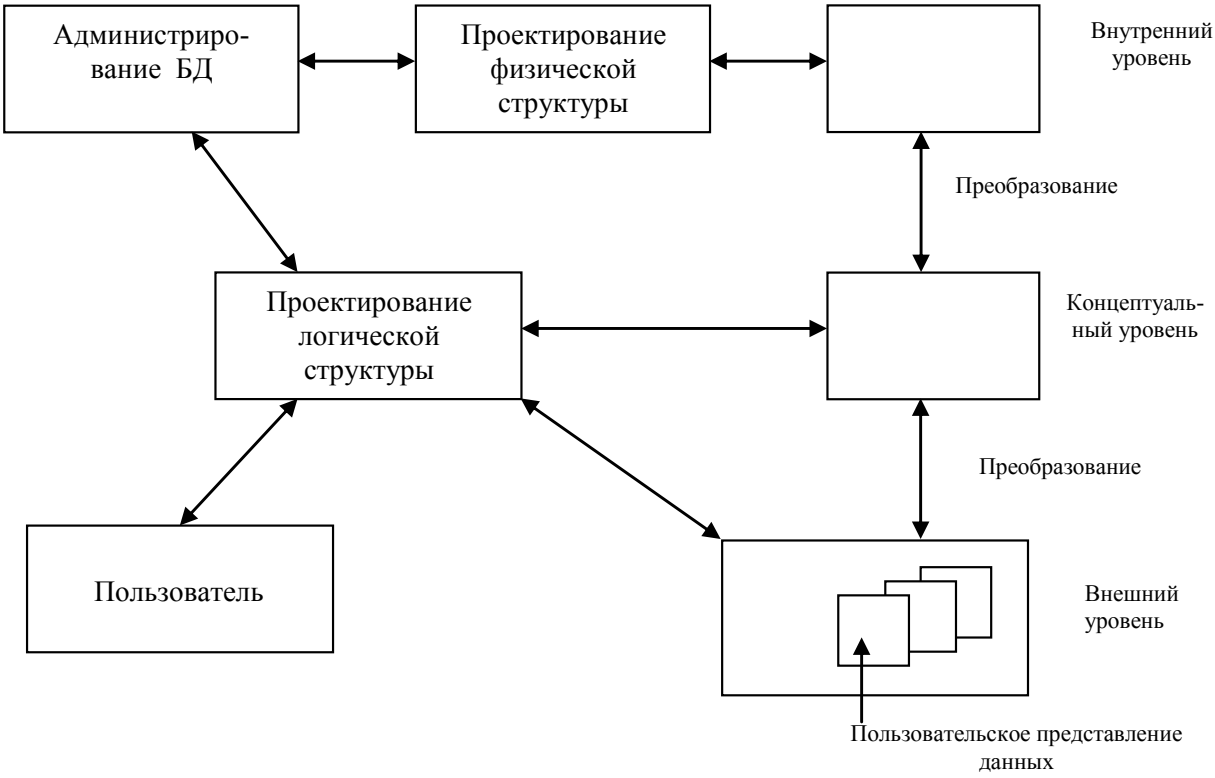


Рис. 2 — Трехуровневая архитектура базы данных
в соответствии со стандартом ANSI/SPARC

Контрольные вопросы

1. Объясните своими словами термины: информационная система, база данных, предметная область.

2. Перечислите и кратко охарактеризуйте каждый компонент современной информационной системы, использующей базу данных.

3. Какие из следующих утверждений могут рассматриваться как данные, какие — как информация?

а) Товар К345-Б выгоден.

б) Товар К345-Б производится фирмой «Диалог».

в) Сергей Иванов в прошлом году получил комиссионных на большую сумму, чем другой торговый агент.

г) Катя Иванова родилась 17 декабря 1988 года.

д) Сегодня стоит хорошая погода.

4. Объясните своими словами понятия: концептуальная модель, внешняя модель.

5. Напишите, чем характерны документальные системы.

6. Объясните, чем концептуальная модель отличается от логической модели.

7. Какие составляющие имеет объектная система.

8. Из каких подсистем состоит АИС?

9. Объясните своими словами понятие банк данных?

10. Какие составляющие имеет информационная система?

2. КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Проектирование БД представляет собой сложный процесс, поскольку необходимо построить модель реального мира таким образом, чтобы ею можно было воспользоваться для решения поставленных задач. Модель должна быть адекватна той предметной области, для которой она создается [2, 3, 4, 7].

Главными элементами концептуальной модели являются объекты и отношения между ними.

Объекты — это сущности (вещи, свойства, явления и т.д.), о которых должна накапливаться информация в разрабатываемой системе. Множество всех объектов, имеющих некоторое общее свойство P, называется **объектным множеством.** Элемент объектного множества называется ***объект-элемент*** или ***экземпляр сущности***.

Рис. 3 — Пример объектного множества

Объектные множества можно разделить на лексические и абстрактные. Элементы лексического объектного множества можно отобразить соответствующими знаками, например буквами алфавита. К лексическим множествам можно отнести такие множества, как ФИО, номер студенческого билета, дата зачисления в ВУЗ и т.д.

СТУДЕНТ в данном примере — это абстрактное объектное множество, его нельзя отобразить знаком или буквами поскольку объект *СТУДЕНТ* не является именем или номером студенческого билета, т.к. имя и номер студенческого билета могут меняться, а студент от этого не изменится. Элементы абстрактных множеств часто представляются внутренними номерами, которые вне системы не имеют смысла. Эти внутренние номера называют

идентификаторами объекта или ключами, поскольку они однозначно определяют экземпляр объектного множества.

Объектные множества могут иметь сложную внутреннюю структуру, которая учитывается при проектировании. Некоторые объектные множества могут содержаться внутри других объектных множеств.

Например. Множество СТУДЕНТ_ФВС содержится в множестве СТУДЕНТ.

а *б*

Рис. 4 — Представление конкретизация/обобщение

Здесь СТУДЕНТ_ФВС является конкретизацией объектного множества СТУДЕНТ, а СТУДЕНТ является обобщением объектного множества СТУДЕНТ_ФВС.

Связи выступают в модели в качестве средства, с помощью которого представляются отношения между объектными множествами, имеющими место в предметной области.

Рис. 5 — Объектное отношение

Построенное объектное отношение представляет собой составное объектное множество: СЕМЕЙНАЯ_ПАРА.

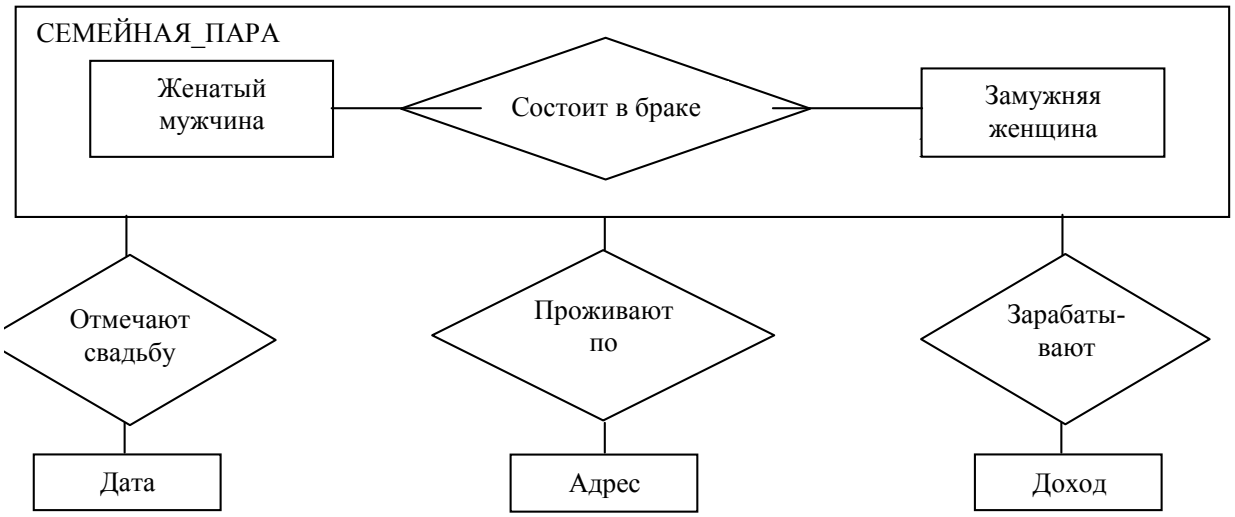


Рис. 6 — Составное объектное множество

При анализе отношений между объектными множествами могут встречаться бинарные (между двумя множествами), тернарные и, в общем случае, n-арные отношения. Наиболее часто встречаются бинарные отношения.

Мощностью называется максимальное количество элементов одного объектного множества, связанных с одним элементом другого объектного множества. Мощности отношений могут быть один_к_одному, один_ко_многим, многие_ко_многим. Отношение, максимальная мощность которого как минимум в одном направлении равна одному, называется ***функциональным.*** В предыдущем примере отношение ***состоит в браке имеет*** мощность один_к_одному. Ниже приведены примеры отношений один_ко_многим и многие_ко_многим.

Рис. 7 — Отношение один_ко_многим

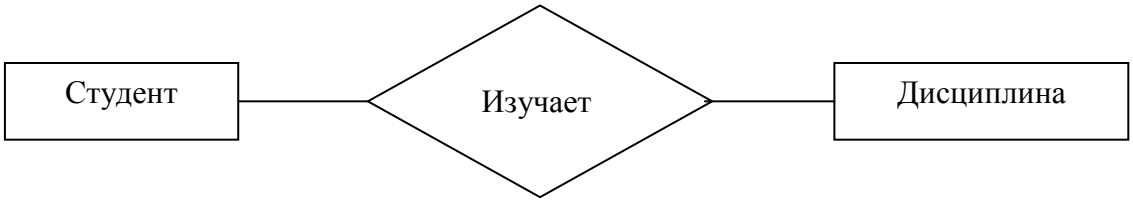


Рис. 8 — Отношение многие_ко_многим

Атрибут — это поименованная характеристика объектного множества, которая принимает значения из некоторого множества значений.

Можно атрибут определить по-другому: функциональное отношение объектного множества к другому объектному множеству. Значение атрибута однозначно определено для каждого элемента объекта. Например, для описания свойств объектного множества КНИГА можно использовать атрибуты: НАЗВАНИЕ, ФАМИЛИЯ_АВТОРА, ЦЕНА, ГОД_ИЗДАНИЯ. Атрибуты могут изменяться, в то время как сам объект остается тем же.

В приведенном выше примере атрибут ЦЕНА может со временем измениться. Может быть пустое значение атрибута, т.е. атрибут не определен. Например, объект КНИГА может не иметь значение атрибута ФАМИЛИЯ_АВТОРА.

Атрибуты, значение которых не меняется, можно использовать в качестве ключей.

Если объект является конкретизацией другого объекта, то он наследует все атрибуты и отношения обобщенного объекта. Необходимо заметить, что конкретизированный объект может иметь свои дополнительные атрибуты.

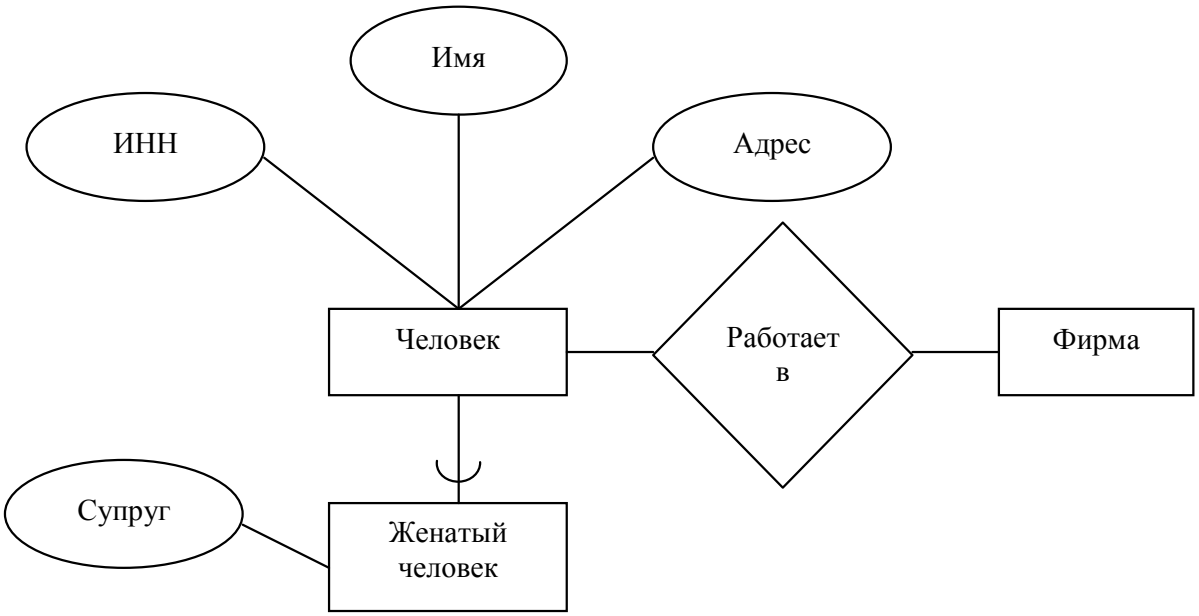


Рис. 9 — Наследование отношений и атрибутов

Конкретный человек Иванов Иван Степанович относится к множеству ЖЕНАТЫЙ ЧЕЛОВЕК, которое является подмножеством объектного множества ЧЕЛОВЕК. Иванов Иван Степанович наследует атрибуты: ИНН, ИМЯ, АДРЕС и отношение РАБОТАЕТ_В. Обладает собственным атрибутом СУПРУГ.

Рассмотрим пример — модель проектной организации.

Проектная организация разрабатывает некие проекты, для которых необходимы определенные детали. Проектная организация связана с рядом поставщиков деталей. В организации несколько отделов. На основании этих данных можно построить следующую модель (рис. 10).

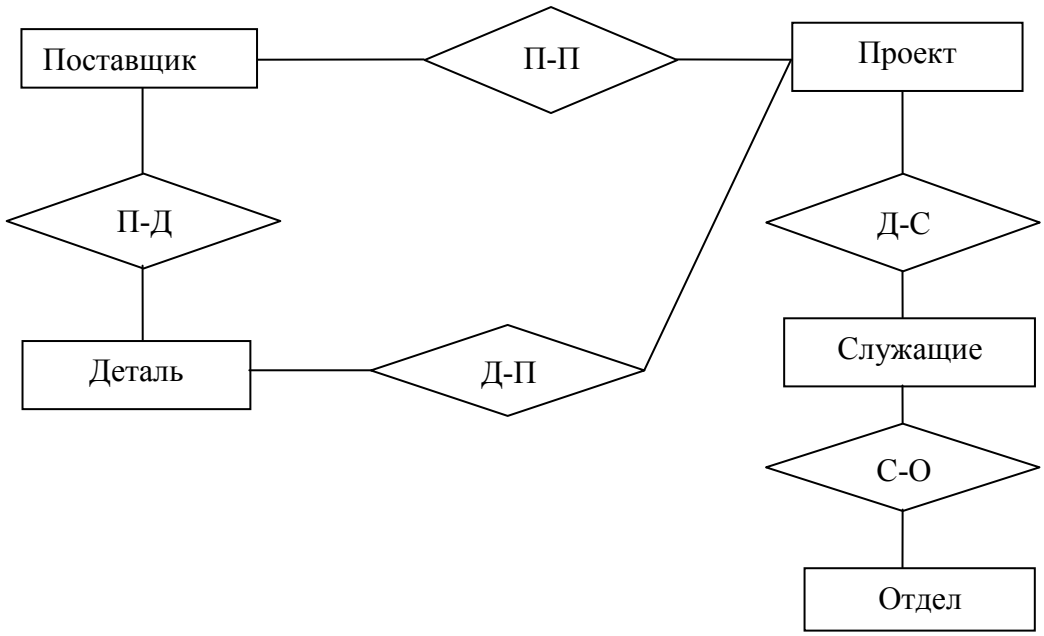


Рис. 10 — Модель предметной области для проектной организации с бинарными связями

Рассмотрим объекты ПОСТАВЩИК, ПРОЕКТ, ДЕТАЛЬ. При построении бинарных отношений ПОСТАВЩИК_ПРОЕКТ (П-П), ПРОЕКТ_ДЕТАЛЬ (Д-П), ПОСТАВЩИК_ДЕТАЛЬ (П-Д) можно ответить на вопросы: какие детали нужны для проекта А1; поставляет ли поставщик АВ детали для проекта А1; поставляет ли поставщик АВ деталь АС. Но если нужно ответить на вопрос: поставляет ли поставщик АВ деталь АС для проекта А1, то исходя из построенной модели на этот вопрос ответить сложно.

В этом случае добавляют тернарную связь ПОСТАВЩИК-ДЕТАЛЬ-ПРОЕКТ (П-Д-П).

Над проектом работают много служащих, но из всей массы служащих нет возможности выделить руководителя проекта. В таких случаях можно добавить дополнительную связь РУКОВОДИТЕЛЬ-ПРОЕКТ (П-РП). Может возникнуть необходимость в получении информации о том, из каких частей состоит полученная деталь. Для получения такой информации следует построить отношение ДЕТАЛЬ-ДЕТАЛЬ (Д-Д). В результате преобразований получим модель, приведенную на рисунке 11.

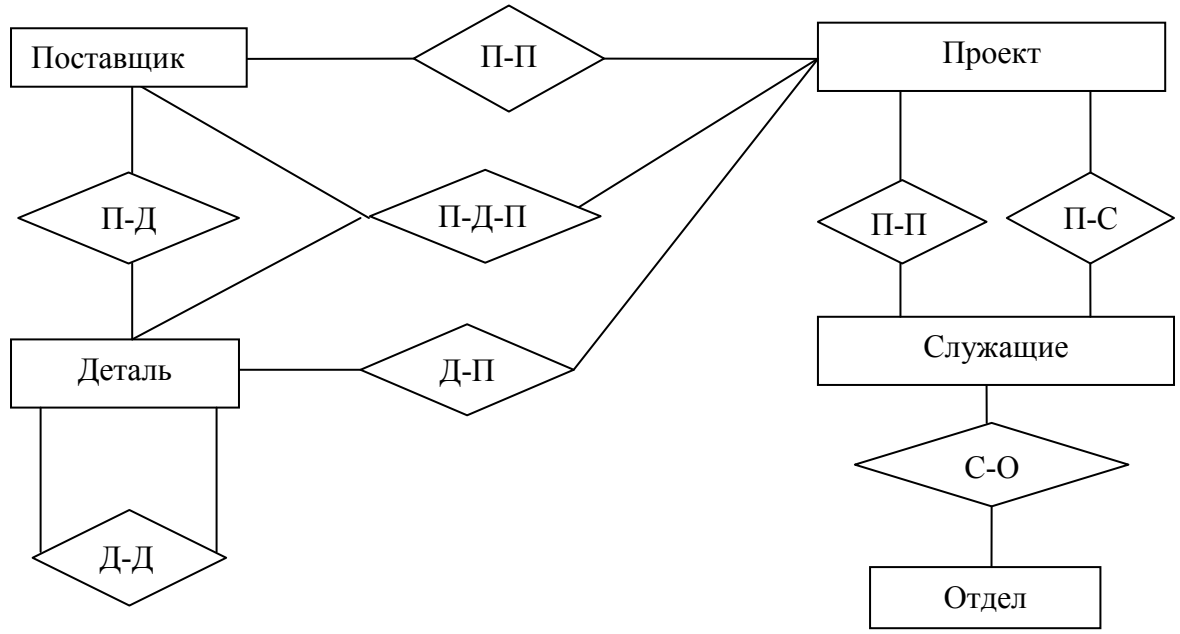


Рис. 11 — Модель с другими типами связей

Рассмотрим более сложные отношения.

Допустим, что необходимо отследить продажу разных видов товаров по фирмам. Для этого построим отношение ТОВАР-ФИРМА, с атрибутом — КОЛИЧЕСТВО (рис. 12). Возникает вопрос: как отследить количество проданного товара? Если КОЛИЧЕСТВО припишем в качестве атрибута ко множеству ТОВАР, в этом случае невозможно отследить, в какую фирму и сколько продано товара (рис. 12, *а*). Если атрибут КОЛИЧЕСТВО припишем фирме, то неизвестно, какой товар продан фирме (рис. 12, *б*). Можно заметить, что атрибут КОЛИЧЕСТВО зависит как от товара, так и от фирмы, в которую продан товар. Поэтому КОЛИЧЕСТВО является атрибутом отношения ТОВАР-ФИРМА (рис. 12, *в*).

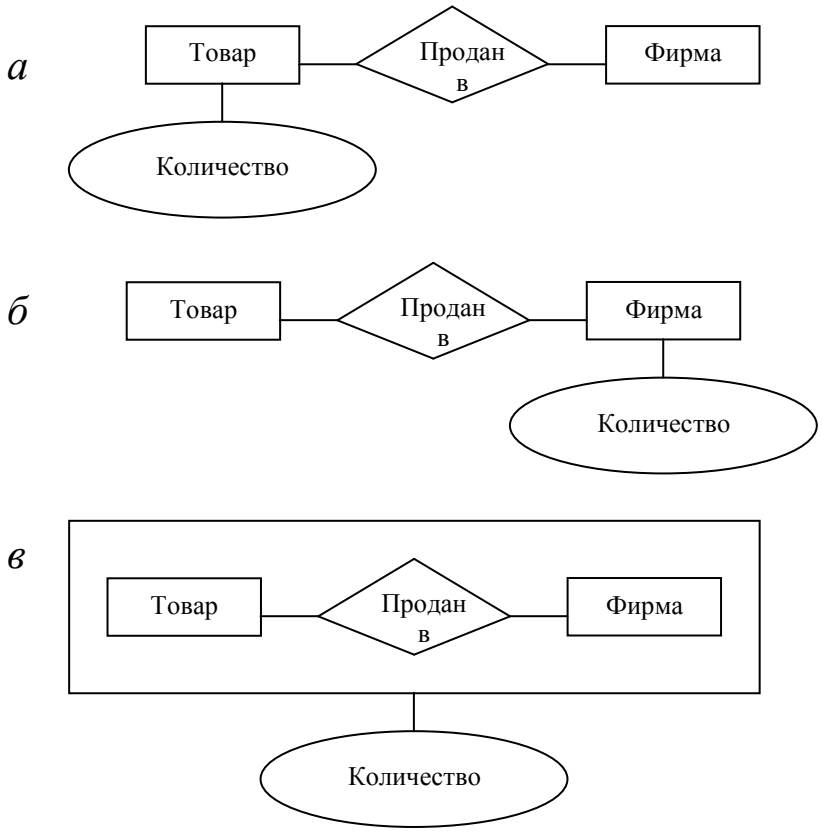


Рис. 12 — Модели отслеживания продаж

В дополнения к вышесказанному рассмотрим пример: модель формирования заказа на товар (рис. 13). Поставщику сделан заказ на поставку товара.

В приведенной модели ЦЕНА, ОПИСАНИЕ, ИНВЕНТАРНЫЙ НОМЕР относятся к объектному множеству ТОВАР.

НОМЕР, НАЛОГ, ИТОГО, ДАТА относятся к объектному множеству ЗАКАЗ. КОЛИЧЕСТВО и СУММА являются атрибутами составного объекта, поскольку зависят и от объекта ТОВАР и от объекта ЗАКАЗ. При этом следует обратить внимание на то, что СУММА — это вычисляемый атрибут.

Концептуальная модель данных создается на основании анализа вопросов, на которые пользователи хотят получить ответы. При проектировании большой базы данных необходимо работать с разными группами пользователей и создавать несколько моделей данных, несколько локальных представлений, которые затем нужно объединить.

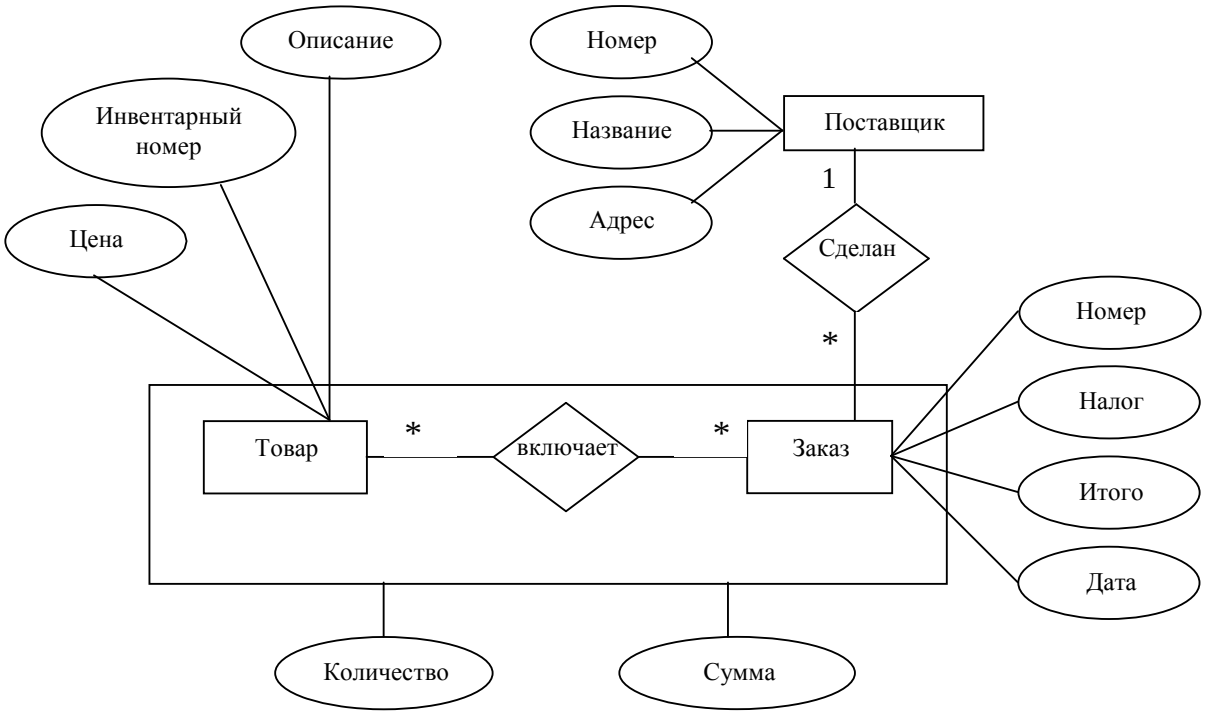


Рис. 13 — Модель формы заказа

Для каждой локальной области необходимо сформулировать сущности, требуемые для ее описания. Сделать это достаточно трудно. Например, выделим две сущности: ДЕТАЛЬ, ИЗДЕЛИЕ и связь — ВХОДИТ В СОСТАВ:

В этом случае, достаточно просто проследить, в какие еще изделия входит деталь. Если эта информация не нужна, то ДЕТАЛЬ можно рассматривать как атрибут сущности ИЗДЕЛИЕ. С какой точки зрения рассматривать предметную область, зависит от того, что конечный пользователь хочет получить на свои запросы к БД.

Объединение локальных представлений носит итеративный характер, в процессе которого выявляются противоречия между локальными представлениями, вызванные некорректностью требований, ошибками, неполнотой спецификаций.

Процесс объединения моделей включает удаление объектных множеств, отношений и атрибутов, которые являются избыточными, и образование новых абстрактных объектных множеств более высокого порядка путем определения новых отношений и продолжается до тех пор, пока не будут устранены все противоречия. В результате проектирования получаем концептуальную модель ПО, оформленную в виде диаграмм.

Данный процесс требует плотного взаимодействия разработчиков и пользователей БД. Следующим этапом проектирования БД является выбор модели данных, которая определяет правила порождения допустимых структур данных, возможные операции над такими структурами, а также классы представимых средствами этой системы ограничений целостности данных.

3. МОДЕЛИ ДАННЫХ

Модель предметной области состоит, как правило, из описания конкретных фактов, а также некоторых общих понятий и закономерностей. Первые образуют базу данных, вторые — базу знаний (БЗ).

К знаниям относят информацию о логике решения задач, а к данным — информацию, которая должна быть проанализирована в соответствии с этой логикой.

Выделяют специфические признаки, отличающие данные от знаний:

1) внутренняя интерпретируемость, т.е. в знаниях находится информация, раскрывающая смысл элементов знаний — информационная и описательная части. Фиксируются все сведения об информационной единице, которые могут понадобиться системе или пользователю для работы с ней;

2) структурированность — свойство декомпозиции сложных объектов в более простые и установление соответствующих связей между ними;

3) связность — в знаниях отражаются закономерности относительно фактов, процессов, явлений и причинно-следственные связи между ними;

4) активность — знания обладают способностью порождения новых знаний, например, при обнаружении неполноты или противоречивости знаний. В качестве информационных единиц, характеризующих некоторое знание, могут выступать присоединенные или встроенные процедуры, что позволяет активизировать эти процедуры в результате появления в базе тех или иных информационных единиц или связей между ними. Это свойство определяется активностью знаний, их первичностью по отношению к процедурам, что не характерно для данных, играющих по отношению к процедурам пассивную роль.

3.1 Понятие модели

В различных системах для задания запроса (например, поискового) из программы пользователя к СУБД применяются разные методы. Можно сказать, что каждая СУБД имеет свои язы-

ковые средства, называемые языком данных (ЯД). При этом необходимо учитывать, что ЯД не язык программирования в традиционном смысле. Язык данных задает правила обработки запроса (например, поиска), а также указывает элементы данных, т.е. ЯД предполагает определенную структуризацию данных (средства описания данных) и средства описания действий, которые необходимо выполнять над данными (средства манипулирования данными). Совокупность средств описания и средств манипулирования называется моделью данных.

Модель знаний содержит набор формальных правил записи знаний (модель данных и семантическое соответствие) и примитивы манипулирования знаниями (правила логического вывода).

В настоящее время широко известны три типа моделей данных: иерархическая, сетевая и реляционная. Наиболее перспективной для дальнейшего развития и применения считается реляционная модель представления данных. Основные исследования всех трех типов моделей данных закончены, и они показали, что все три типа равномощны.

Необходимо заметить, что появилась еще одна модель данных — объектно-ориентированная.

3.2 Иерархические модели

Иерархическая модель данных возникла из практики. Впервые СУБД, использующие эту модель данных, были созданы в 1960-х годах для поддержки лунного проекта Аполлон. Основной посылкой создания стала необходимость управления миллионами деталей, связанных друг с другом иерархическим образом. Иерархические модели удобны и используются для построения рубрикаторов и классификаторов. В основе модели лежит граф типа дерево.

Граф — это объект, состоящий из двух множеств: множества вершин $X=\{x_1, x_2,...,x_3\}$ и множества линий, соединяющих эти вершины, называемых множеством ребер или дуг.

Применительно к модели данных вершины используются для интерпретации сущностей, а линии — для интерпретации связей между типами сущностей.

Древовидная структура или граф типа дерево— это связный граф, не имеющий циклов. Обычно выделяют вершину, в которую не заходит ни одна линия, в этом случае граф становится ориентированным.

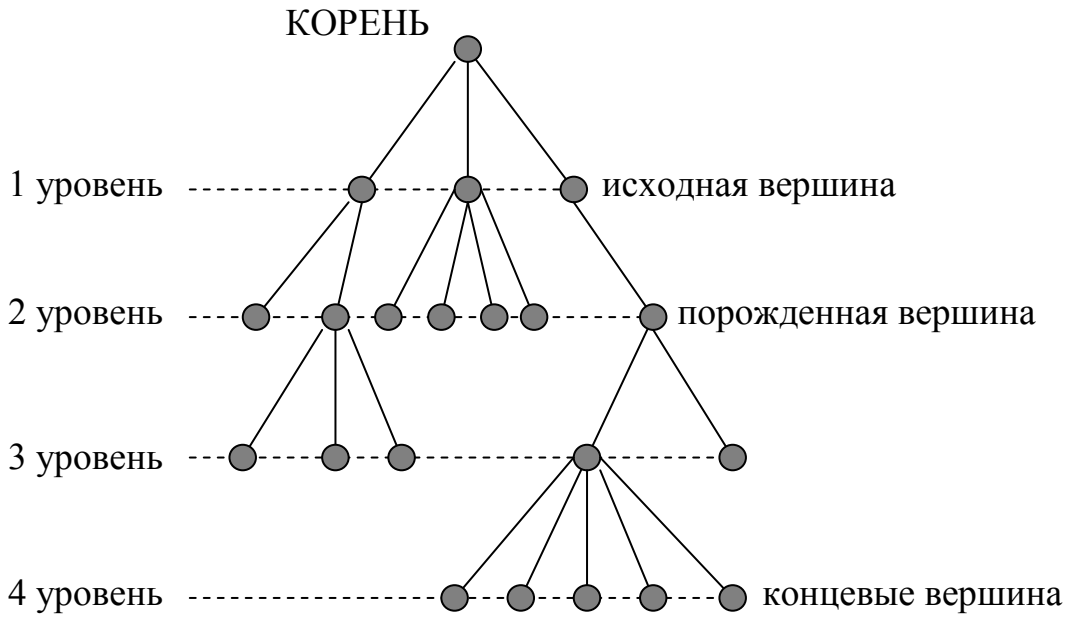


Рис. 14 — Иерархическая структура

Иерархическая структура удовлетворяет следующим условиям:

1. Иерархия начинается с корневой вершины.
2. Каждая вершина соответствует одному или нескольким атрибутам.
3. На уровнях с большим номером находятся порожденные вершины.
4. Доступ к каждой вершине происходит через корневую по единственному пути.
5. Каждая вершина, находящаяся на i-м уровне, связана только с одной вершиной (i-1)-го уровня.
6. Корневая вершина связана с одной или несколькими порожденными (зависимыми) вершинами.
7. Существует произвольное количество вершин каждого уровня.

В терминологии иерархической модели существует несколько основных понятий:

– ***Тип сегмента*** — это поименованная совокупность типов элементов данных, в него входящих, соответствует объекту.

– ***Экземпляр сегмента*** образуется из конкретных значений полей или сегментов данных, в него входящих.

Набор всех экземпляров сегментов, подчиненных одному экземпляру корневого сегмента, называется физической записью.

Количество экземпляров потомков может быть различным для разных экземпляров родительских сегментов, поэтому в общем случае физические записи имеют различную длину.

На рисунках 14, 15 представлена иерархическая структура, соответствующая отношениям в ВУЗе. В ВУЗе несколько факультетов. В примере на рисунке 15 их указано четыре. В реальной жизни их может быть другое количество. Каждый факультет «содержит» разное количество кафедр.

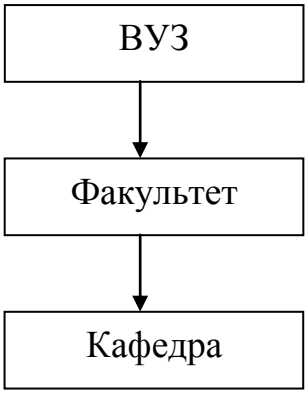


Рис. 15

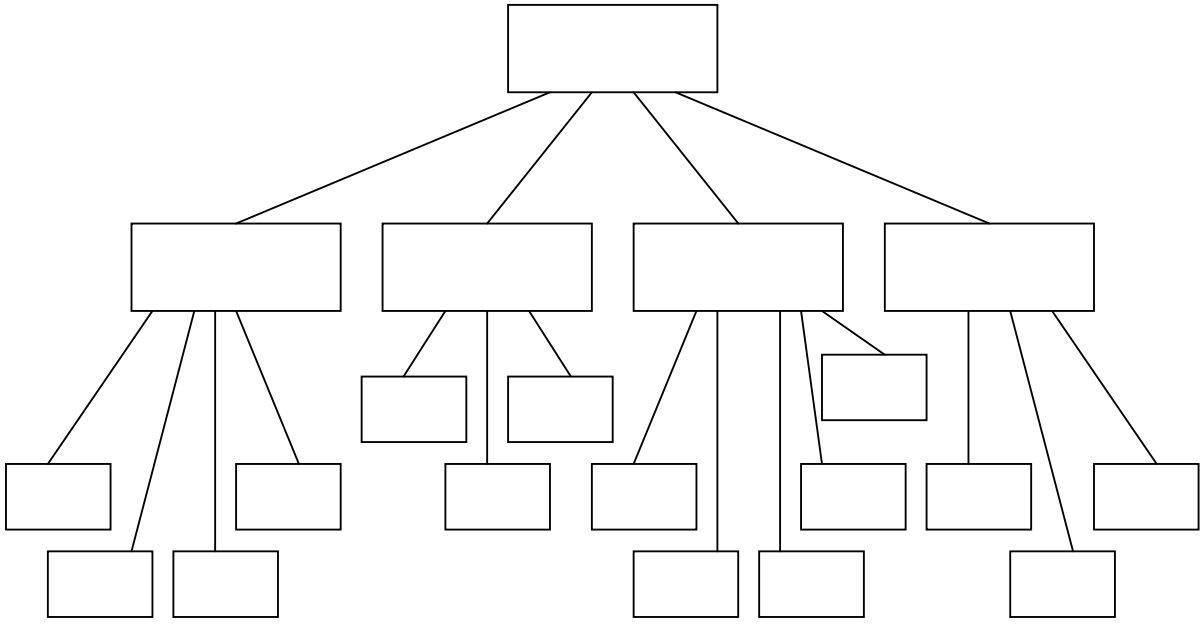


Рис. 16

Неудобство такой системы представления данных заключается в том, что реальный мир не может быть легко представлен в виде древовидной структуры с единственным корнем. Примерно в это же время сформировалась другая модель данных — сетевая.

3.3 Сетевые модели

Сетевые модели данных также базируются на использовании графовой модели представления информации. Вершины соответствуют типам, дуги — связям типов.

В сетевой модели допускаются произвольные связи между сущностями. Одно из основных достижений сетевой модели данных состоит в том, что впервые были введены языки представления данных (Data Definition Language, DDL) и языки манипулирования данными (Data Manipulation Language, DML).

При реализации сетевой модели можно применять различные представления данных, но основой описания является понятие набора.

Кроме понятия набора данных в сетевых БД применяется понятия: элемент данных, агрегат данных и запись. ***Элемент данных*** **—** это минимальная единица информации, доступная пользователю с использованием СУБД. Агрегат данных соответствует следующему уровню обобщения модели и представляет собой вектор или повторяющуюся группу. Совокупность агрегатов или элементов данных образует запись.

Набор представляет собой поименованное двухуровневое дерево. Исходная запись (1-й уровень) называется владельцем набора, а порожденные записи — членами этого набора. Каждый из членов одного набора может быть объявлен владельцем другого набора. Фактически наличие подобных возможностей позволяет промоделировать отношения МНОГИЕ_КО_МНОГИМ. Таким образом, можно описать достаточно сложную сетевую структуру.

Среди всех наборов данных выделяю специальный тип набора, называемый «Сингулярным набором», владельцем которого формально определена вся система.

Принципиальное значение в сетевой модели имеет то положение, что предусматривается обработка (одновременная обработка) только одиночных объектов из базы данных.

Механизмы доступа к данным и навигация сложны, так как необходимо, двигаясь по структуре:

а) найти нужную запись;

б) сделать ее текущей;

в) извлечь в рабочий буфер;

г) изменить данные;

д) записать содержимое буфера в БД.

Возможен переход от представления одной структуры данных к другой. Приёмы просты, но приводят к избыточности в логическом представлении данных.

3.4 Реляционные модели данных

Теоретической основой этой модели данных стала теория отношений. Было показано, что множество отношений замкнуто относительно некоторых специальных операций, т.е. образует вместе с этими операциями абстрактную алгебру.

Реляционная модель была предложена в 1970 году Коддом. Основу реляционной модели данных составляет совокупность данных, сформированных в виде таблицы. Такая форма привычна для специалистов, пользующихся различного рода справочной литературой. Формальным аналогом таблицы выступает отношение.

Пусть дана совокупность множеств D_1, D_2, $...D_n$. Декартово произведение $D_1 \times D_2 \times ... \times D_n$ — множество всех возможных ***кортежей*** $<d_1,d_2,... d_n>$ таких, что $d_i \in D_i$, $i = 1,...,n$.

Множества D_i называют ***доменами***.

N-арным отношением R называется некоторое подмножество декартова произведения этих множеств: $R \subseteq D_1 \times D_2 \times ... \times D_n$.

Пример

Пусть заданы множества $D_1 = \{a_1,a_2,a_3,a_4,a_5\}$ и $D_2=\{b_1,b_2,b_3\}$, тогда декартово произведение этих множеств запишется как

$D_1 \times D_2 = \{<a_1,b_1>,<a_1,b_2>,<a_1,b_3>,<a_2,b_1>,...,<a_5,b_3>\}$.

Построим отношение

$R=\{<a_1,b_3>,<a_2,b_1>,<a_2,b_3>,<a_3,b_1>,<a_3,b_3>\}$.

Очевидно, что $R \subset D_1 \times D_2$.

Совокупность кортежей, записанных друг за другом, образует таблицу, строки которой соответствуют кортежам, а столбцы — атрибутам.

A_1	A_2
a_1	b_3
a_2	b_1
a_2	b_3
a_3	b_1
a_3	b_3

Атрибут A_i представляет собой подмножество домена D_i. Для предыдущего примера атрибутами являются множества $A_1 = \{a_1, a_2, a_3\}$, $A_2 = \{b_1, b_3\}$, при этом $A_1 \subset D_1$ и $A_2 \subset D_2$.

Запись: $R(A_1, A_2, \ldots, A_n)$ называется ***схемой отношения*** R и содержит наряду с названием отношения названия атрибутов. Если атрибуты принимают значения из одного и того же домена, то они называются ***Θ-сравнимыми***, где Θ — множество допустимых операций сравнения, заданных для данного домена. Схемы двух отношений называются ***эквивалентными***, если они имеют одинаковую степень и возможно такое упорядочение имен атрибутов в схемах, что на одинаковых местах будут располагаться сравнимые атрибуты, то есть атрибуты, принимающие значения из одного домена.

Поскольку отношение — это подмножество, то следует отметить, что оно не может содержать двух одинаковых кортежей.

Подмножество $K = \{A_{i1}, A_{i2}, \ldots, A_{ik}\}$ заданной схемы отношения $R(A_1, A_2, \ldots, A_n)$ будем называть ключом, если в двух различных кортежах отношения R значения в каждом атрибуте из K не все одинаковы и ни одно собственное подмножество K этим же свойством не обладает.

Другими словами, ключ — это такое множество атрибутов, которое при задании этих атрибутов однозначно определяет кортеж.

Отношение (в БД) моделирует реальную ситуацию, и в соответствии со свойствами отношений в таблице могут присутствовать только атрибуты, обладающие свойством симметричности.

В результате вышесказанного реляционная таблица обладает рядом специфических свойств:

1. В таблице нет двух одинаковых строк.
2. Таблица имеет столбцы, соответствующие атрибутам отношения.
3. Каждый атрибут в отношении имеет уникальное имя.
4. Порядок строк в таблице произвольный.

5. Два отношения, отличающиеся только порядком следования столбцов, интерпретируются как одинаковые.

Операции над отношениями выполняются методами реляционного исчисления и реляционной алгебры. Кодд показал, что реляционная алгебра и реляционное исчисление — эквивалентны.

3.4.1 Реляционная алгебра

Множество M вместе с набором операций

$$\sum=\{\varphi_1,...,\varphi_m\}, \quad \varphi_i : M^{n_i} \to M,$$

где n_i — арность операции φ_i, называется алгебраической структурой, универсальной алгеброй или просто ***алгеброй.*** Множество M называется ***основным*** (***несущим множеством***), множество операций $\sum$ называется ***сигнатурой***.

Основным множеством в реляционной алгебре является множество отношений. Поскольку отношение можно представить в виде таблицы, то здесь в качестве операндов будем рассматривать реляционные таблицы. Пусть заданы отношения *R* и *S*, которые содержат перечни изделий, выпускаемых разными фирмами F_1 и F_2 соответственно.

S	
Prod_id	Prod
1035	Свитер
2241	Настольная лампа
2249	Светильник
2378	Бронзовая скульптура

R	
Prod_id	Prod
1055	Свингер
2246	Люстра
2249	Светильник

Основные операции реляционной алгебры: сумма, пересечение, вычитание, произведение, выбор, проекция, Θ — соединение, деление. Первые четыре взяты из теории множеств, следующие применяются в реляционной модели данных.

Объединением двух отношений *R* и *S*:$R \cup S$ называется множество всех кортежей, содержащихся в *R* и *S*.

В результате операции объединения получим следующую таблицу:

Q	
Prod_id	Prod
1035	Свитер
2241	Настольная лампа
1055	Свингер
2246	Люстра
2249	Светильник
2378	Бронзовая скульптура

Необходимо отметить, что объединяются таблицы только с одинаковыми атрибутами и кортежи одинаковые для обеих таблиц, входят в результирующую таблицу *Q* только один раз.

Пересечением $R \cap S$ отношений *R* и *S* является множество кортежей, одновременно содержащихся в отношениях *R* и *S*.

Q	
Prod_id	Prod
2249	Светильник

В этом случае в отношении *Q* содержится перечень изделий, выпускаемых обеими фирмами.

Разностью *R**S* отношений *R* и *S* называется отношение, содержащее множество всех кортежей, содержащихся в *R* и не содержащих кортежи из *S*.

В результате выполнения операции ***разность*** получим отношение *Q*, содержащее перечень изделий, выпускаемых только фирмой F_2.

*R**S*

Q	
Prod_id	Prod
1055	Свингер
2246	Люстра

Необходимо отметить, что для получения перечня изделий, выпускаемых только фирмой F_1, необходимо выполнить операцию *S**R*. В этом случае получаем отношение Q_1 вида:

Q_1	
Prod_id	Prod
1035	Свингер
2241	Настольная лампа
2378	Бронзовая скульптура

Декартовым произведением *R*×*S* отношений *R* и *S* является множество кортежей $t = <r_1, r_2,\ldots,r_m, S_1, S_2,\ldots, S_n>$, где первые *m* элементов кортежа *t* образуют кортеж из множества *R*, последние

n элементов из отношения S, т.е. $<r_1, r_2, \ldots, r_m> \in R$ и $<S_1, S_2, \ldots, S_n> \in S$. Рассмотрим две таблицы A и B.

A	
id	*N*
10	27
11	35

B	
Цвет	Статус
Зеленый	22
Красный	17
Сиреневый	54

В результате выполнения операции произведения получим таблицу $T=A\times B$ вида

T			
id	*N*	Цвет	Статус
10	27	Зеленый	22
10	27	Красный	17
10	27	Сиреневый	54
11	35	Зеленый	22
11	35	Красный	17
11	35	Сиреневый	54

Произведение создано путем присоединения к каждой строке таблицы *A* каждой строки таблицы *B*.

Выборкой $\alpha_F(R)$ из отношения *R* является множество кортежей отношения *R*, для которых истинен предикат *F*.

Предикат — это некоторое условие, выраженное в форме логического утверждения. Условия в основном такие же, какие используются в операторе IF в выражениях традиционных языков программирования. В качестве операндов здесь выступают столбцы реляционной таблицы, используются арифметические операторы сравнения $<$, $\leq$, $=$, $\geq$, $\neq$, $>$ и логические операторы ^ (И), | (ИЛИ), ¬ (НЕ). В выражении условия могут присутствовать скобки.

В результате выполнения операции «выборка» из реляционной таблицы получим реляционную таблицу, все кортежи которой будут удовлетворять заданному условию.

Пусть существует реляционная таблица, в которой перечислены фамилии студентов 1 курса и их оценки по предметам «Математика», «Физика», «Информатика». Необходимо найти всех

студентов, которые сдали эти предметы на «5». Условное выражение в этом случае запишется как

(«Математика»=5)^(«Физика»=5)^(«Информатика»=5)

Пусть R (A_1, A_2,…, A_n) — n-арное отношение.

Проекцией $\Pi_x(R)$ отношения R на множество доменов $X=\{A_{i1}, \ldots, A_{ik}\}$ называется такое множество всех кортежей $t = <v_1, v_2,\ldots, v_k>$, что существует кортеж $r = <r_1, \ldots, r_m>$ отношения k, в котором $r_{ij} = v_j$, j =1, …, k. Операция «проекция» позволяет исключить из таблицы ненужные столбцы, т.е. во вновь создаваемую реляционную таблицу выбираются только необходимые столбцы. Необходимо отметить, что операция создания проекций автоматически удаляет повторяющиеся кортежи из результирующей таблицы.

Операция соединения используется для связывания данных между таблицами. Новое отношение строится посредством конкатенации кортежей двух исходных отношений. У этой операции есть несколько версий: естественное соединение, тета-соединение и внешнее соединение.

При естественном соединении связываются таблицы, когда общие столбцы имеют равные значения.

Пусть отношение A и B имеют следующие схемы отношения соответственно.

$$A = \{x_1, x_2,\ldots, x_m \quad y_1,\ldots, y_n\},$$
$$\text{B} = \{y_1,\ldots, y_n \quad z_1,\ldots z_p\}.$$

Естественным соединением называется отношение с заголовком $\{x, y, z\}$ и телом, содержащим множество всех кортежей $\{X{:}x, Y{:}y, Z{:}z\}$ ткаих, что в отношении **A** значение атрибута X равно x, значение атрибута Y равно y, а в отношении B значение атрибута Y равно y, а атрибута Z равно z.

R	
a_1	b_1
a_2	b_2
a_3	b_3

Q	
b_1	c_1
b_2	c_2
b_3	c_3

В результате применения операции естественного соединения получим таблицу T.

T		
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

Тета-соединение — это соединение с определенным условием, в котором участвуют столбцы из каждой таблицы. Например, получить P соединением A и B при условии L. Это означает, что два столбца будут определенным образом сравниваться, при этом может использоваться один из операторов сравнения =, ≠, <, >, ≤, ≥. То есть связываются таблицы, когда значения из определенных столбцов находятся в определенном отношении.

Внешнее соединение расширяет естественное соединение. Сначала выполняется естественное соединение, затем, если какая-либо строка одной из исходных таблиц не подходит ни к одной строке второй таблицы, она включается в таблицу соединения, а все дополнительные столбцы заполняются пустыми значениями.

R	
a_1	b_1
a_2	b_3
a_3	b_4

Q	
b_1	c_1
b_2	c_2
b_4	c_3

В результате применения операции внешнего соединения получим таблицу T_1.

T_1		
a_1	b_1	c_1
a_3	b_4	c_3
a_2	b_3	–
–	b_2	c_2

Операция деления создает новую таблицу путем выбора строк одной таблицы, соответствующих каждой строке другой таблицы. Является обратной операции декартова произведения. Для нее выполняется условие: $(R*S)/S=R$.

3.4.2 Реляционное исчисление

Реляционное исчисление базируется на теоретических основах исчисления предикатов. Использование реляционного исчисления дает возможность манипулировать с данными на уровне выходного документа, что позволяет строить удобные для пользователя языки манипулирования данными непроцедурного типа.

Логика предикатов является компонентом естественного языка и содержит те составные его части, которые не содержат внутри себя нечеткостей. Напомним, что ***предикат*** — это функция, аргументы которой принимают значения из предметной области M, и значение которой суть 0 или 1.

$F^{(n)}(\ldots)$: $M^n \to \{0,1\}$, $M^n = M \times M \times \ldots \times M$.

Введем алфавит, используемый в языке предикатов.

Сущность обозначают латинскими буквами a, b, c,…x, y, z. Множество, о котором будет идти речь, называется ***предметной областью*** и обозначается M. a, b, c,…x, y, z $\in$M Константные элементы из M обозначаются a, b, c. Буквами ближе к концу алфавита обозначаются переменные.

Для обозначения предикатов будем использовать функциональные символы, в скобках через запятую — последовательность констант и переменных: $F^{(2)}(x,y)$, $S^{(1)}(a)$, $H^{(3)}(x,y,z)$, индекс обозначает местность предиката.

Определим предикат <, он запишется — $F^{(2)}(x,y)$, где x, y $\in$M и M={1,2,…,9}.

Значение функции, константы и переменные объединяются одним названием — терм. При построении высказываний используются логические связи $\wedge$ — коньюнкция (И), $\vee$ — дизьюнкция (ИЛИ), $\neg$ — отрицание (НЕ), $\to$ импликация (ЕСЛИ…, ТО…), n — эквивалентность (ЕСЛИ И ТОЛЬКО ЕСЛИ), $\otimes$ — дизьюнкция с исключением (ИЛИ…, ИЛИ…).

Таблица 1 — Таблица истинностей для операций

a	b	a$\wedge$b	A$\vee$b	a$\otimes$b	a$\to$b	a~b
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	0
1	1	1	1	0	1	1

Кроме того, применяются термы сравнения, имеющие общий вид $x_i \circ x_j$, где $\circ$ — символ операции сравнения: =, $\neq$, <, >, $\leq$, $\geq$.

Кванторы существования $\exists$ и всеобщности $\forall$ позволяют отнести высказывание ко всему рассматриваемому множеству.

Рассмотрим произвольный одноместный предикат F(x). Выражение (∃x)F(x) означает, что существует x, для которого F(x) истинно.

$$(\exists x)F(x) = \begin{cases} 0, \text{ если при любом x } F(x) \text{ ложно;} \\ 1, \text{ если существует хотя бы один x, при котором} \\ \quad \text{высказывание } F(x) \text{ — истина.} \end{cases}$$

Выражение (∀x) F(x) означает, что для всех x F(x) истинен.

$$(\forall x)\ F(x) = \begin{cases} 0, \text{ если есть хотя бы один } x \in M, \text{ при котором } F(x)=0; \\ 1, \text{ если для всех } x \in M,\ F(x) = 1. \end{cases}$$

По определению кванторов всеобщности и существования имеем следующие соотношения:

$$(\exists x)P(x) \leftrightarrow (P(b_1) \vee P(b_2) \vee \ldots \vee P(b_m));$$
$$(\forall x)P(x) \leftrightarrow (P(b_1) \wedge P(b_2) \wedge \ldots \wedge P(b_m));$$

где $\{b_1, b_2, \ldots, b_m\} = X$ — область определения аргумента x.

Реляционное исчисление и предикаты

В реляционном исчислении принято связывать с отношением $R(A_1, A_2, \ldots, A_n)$ некоторый предикат $P(x_1, x_2, \ldots, x_n)$, при этом аргументы (отношения и предиката) имеют одинаковые области определения таким образом, что, если $P(a_1, a_2, \ldots, a_n) = 1$, то кортеж $(a_1, a_2, \ldots, a_n)$ принадлежит отношению R, $a_i \in A$, $i = 1, \ldots, n$.

В противном случае кортеж не входит в состав указанного отношения. Отсюда следует, что посредством задания некоторого предиката может быть задано и соответствующее ему отношение.

На основании предикатов, операций над ними и правил строятся правильные формулы.

Реляционное исчисление является одним из видов исчисления предикатов, используемого в качестве основы для создания языка запросов к базе данных. Существует реляционное исчисление кортежей и доменов. В реляционном исчислении кортежей запрос имеет вид: $\{x \mid f(x)\}$, где x — переменная для кортежей;

f — формула исчисления. Смысл запроса состоит в том, чтобы найти множество кортежей, для которых формула $f(x)$ — истина.

Операции делятся на две большие группы: поиск (запрос) и обновление данных. Например, запросы к базе данных выражаются в виде формул:

сумма $R \cup S: \{T | R(T) \vee S(T)\}$;

произведение $R \cap S: \{T | R(T) \wedge S(T)\}$;

разность $R - S: \{T | R(T) \wedge \neg S(T)\}$.

Если вместо значений переменных для кортежей возьмем значения переменных для доменов? получим реляционное исчисление доменов.

3.5 Объектно-ориентированные БД

При всей эффективности и успехе развития реляционных БД, существуют некоторые приложения, которые в результате своей сложности плохо реализуются перечисленными выше моделями данных. Объектно-ориентированные СУБД (ООСУБД) позволяют обрабатывать сложные объекты, содержат наследование и другие свойства, что делает возможным реализацию объектно-ориентированных концептуальных моделей.

Объектно-ориентированный подход достаточно прост. Прежде всего вспомним, что в информатике необходимо обеспечивать тесные связи между информационными системами и системами из реального мира которые они представляют. Во многих моделях свойственна потеря семантики, поэтому всегда существует риск, что разработанная система «неправильно построена». Она либо встречает сопротивление пользователей, либо не может решить поставленную задачу. Обычно объекты реального мира обладает некоторыми свойствами и применимыми к ним функциями, которые обеспечивают их определение и классификацию. Объекты связываются друг с другом либо на межобъектной основе в соответствии с бизнес правилами, либо с помощью некоторого вида наследования свойств. Далее, к объектам реального мира применяются некоторые операции и функции. Когда эти объекты моделируются и транслируются в структуры данных, например, в реляционную, имеет место потеря семантики. Даже

при реализации моделей способных «удерживать» семантику, например, «сущность-связь» — результаты не обнадеживающие. Дело в том, что сами системы БД не поддерживают семантику внутренними средствами.

При разработке ООБД необходимо было создание структур, которые могли учитывать специфику приложений и удерживать семантику. Концепции ООБД:

Индивидуальность объектов. Каждый объект в объектно-ориентированной (ОО) системе имеет индивидуальный идентификатор, обычно называемый ID объекта или IDentificator. Использование ID позволяет изменять значение любого атрибута объекта, в том числе и атрибутов, образующих первичный ключ, не нарушая при этом ссылок на данный объект. Концепция ID берет начало от реляционных БД.

Атрибуты. Данный объект всегда имеет два аспекта: состояние и поведение. Состояние определяется множеством значений его атрибутов (или свойств, или экземплярных переменных, или полей). В ОО средах значение любого заданного атрибута должно подчиняться некоторым правилам, связанным с типом данных, с диапазоном или списком значений, а также с характеристикой, указывающей, единственное или множественное значение может быть задано и т.д.

Методы. Методы описывают поведение объекта. Благодаря передачи явных сообщений между объектами системы, вызов инкапсулированных методов и доступ к атрибутам заданного объекта осуществляется в соответствии с правилами данной ООСУБД.

Классы. Классы, называемые также типами, являются инструкциями, которые представляют собой группы объектов обладающих одним и тем же множеством атрибутов и методов. Классы могут быть примитивными, в этом случае не иметь атрибутов. Они могут иметь атрибуты и методы и в этом случае все объекты, относящиеся к данному классу имеют одни и те же множества атрибутов и объектов.

Иерархии классов и наследование. Подклассы наследуют атрибуты суперкласса и имеют некоторые новые атрибуты, специфические для принадлежащих им объектов.

Долговременное хранение. Корни ОО БД лежат в языках программирования. Создание нового долговременно хранимого объекта в языке программирования, порождает объект БД, который может использоваться непосредственно в программе без необходимости отображать его в структуры памяти языка программирования.

4. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

Существуют два основных подхода к моделированию предметной области при проектировании БД.

Первый подход, называемый функциональным, требует учитывать все особенности приложения, предназначенного для работы с проектируемой базой данных. Сначала составляется подробный список всех основных операций приложения. На основе списка задач определяются необходимые таблицы и их структуры. Использование Функционального подхода позволяет максимально оптимизировать структуру БД для выполнения указанного перечня задач. Такая структура в дальнейшем менее пригодна для изменения.

Второй подход объектный предлагает рассматривать предметную область без непосредственного учета требований приложений, которые будут работать с БД. При разработке структуры необходимо выделить в предметной области наиболее важные объекты. Сопоставить им отдельную таблицу. Использование второго подхода обеспечивает минимальное количество ошибок проектирования и зафиксировать реальные связи между объектами предметной области. Оба подхода используются при проектировании БД.

4.1 Выбор ключевых полей

Рассмотрим шаги проектирования. Взглянем на фрагмент таблицы Customers (клиенты) из базы данных NorthWind. Она выглядит следующим образом:

CustomerID	CompanyName	City	Country
ALFKI	Alfreds Futterkiste	Berlin	Germany
ANATR	Ana Trujillo Emparedados y helados	Mexico D.F.	Mexico
ANTION	Antonio Moreno Taqueria	Mexico D.F.	Mexico
AROUT	Around the Horn	London	UK

Поскольку строки в таблице не упорядочены, нам нужна колонка (или набор из нескольких колонок) для уникальной идентификации каждой строки. Такая колонка (или набор колонок) называется *первичным ключом* (*primary key*). Первичный ключ любой таблицы обязан содержать уникальные непустые значения для каждой строки.

Выбор первичного ключа задача очень важная. Часто в проектируемых БД содержащих сведения о клиентах, в качестве первичного ключа выбирают фамилию имя отчество, что является неверным, поскольку эти три поля достаточно часто совпадают и при вводе фамилий чаще всего возникают ошибки ввода. Если в качестве идентификатора использовать наименование компании, то это тоже чревато неприятностями поскольку, например, наименование компании AT & T, A.T. and T, Ma Bell с точки зрения человека это одна компания, а с точки зрения компьютера разные. Чаще всего используют специальный дополнительный столбец, который и будет использоваться в качестве первичного ключа.

Если первичный ключ состоит из более чем одной колонки, он называется *составным первичным ключом* (*composite primary key*). Типичная база данных обычно состоит из нескольких связанных таблиц. Ниже показан фрагмент таблицы Orders (заказы):

OrderID	CustomerID	OrderDate	Freight	ShipAddress
10254	CHOPS	11.07.96	22.98	Hauptstr.31
10259	CENTC	18.07.96	3.25	Sierras de Granada 9993
10265	BLONP	25.07.96	55.28	24,place Kleber
10278	BERGS	12.08.96	92.69	Berguvsvagen 8
10280	BERGS	14.08.96	8.98	Berguvsvagen 8
...	...	...	...	...

Поле CustomerID этой таблицы содержит идентификатор клиента, разместившего данный заказ. Если нам нужно узнать, как называется компания, разместившая заказ, мы должны поискать это же значение идентификатора клиента в поле CustomerID таблицы Customers и в найденной строке прочесть значение поля CompanyName. Иными словами, нам нужно связать две таблицы, Customers и Orders, по полю CustomerID. Колонка, указывающая на запись в другой таблице, связанную с данной записью, называется

внешним ключом (*foreign key*). Как видим, в случае таблицы Orders внешним ключом является колонка CustomerlD (рис. 16).

Иначе говоря, внешний ключ — это колонка или набор колонок, чьи значения совпадают с имеющимися значениями первичного ключа другой таблицы.

Подобное взаимоотношение между таблицами называется связью (*relationship*). Связь между двумя таблицами устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой.

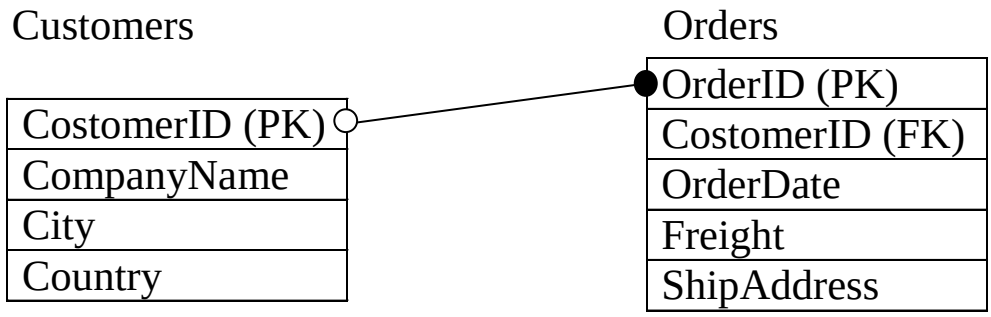


Рис. 17 — Первичные и внешние ключи в таблицах Customers и Orders

Если каждый клиент в таблице Customers может разместить только один заказ, говорят, что эти две таблицы связаны соотношением один-к-одному (*one-to-one relationship*). Если же каждый клиент в таблице Customers может разместить ноль, один или много заказов, говорят, что эти две таблицы связаны соотношением *один-ко-многим* (*one-to-many relationship*) или соотношением *master-detail.* Подобные соотношения между таблицами используются наиболее часто. В этом случае таблица, содержащая внешний ключ, называется *detail-таблицей,* а таблица, содержащая первичный ключ, определяющий возможные значения внешнего ключа, называется *master-таблицей.* В нашем примере таблица Orders содержит внешний ключ CostomerID, который является внешним ключом. Таблица Customers является *master-таблицей,* а таблица Orders — *detail-таблицей.*

Группа связанных таблиц называется *схемой* базы данных (*database schema*). Информация о таблицах, их колонках (имена,

тип данных, длина поля), первичных и внешних ключах, а также иных объектах базы данных называется *метаданными* (*metadata*).

Любые манипуляции с данными в базах данных, такие как выбор, вставка, удаление, обновление данных, изменение или выбор метаданных, называются запросом к базе данных (*query*). Обычно запросы формулируются на каком-либо языке, который может быть как стандартным для разных СУБД, так и зависящим от конкретной СУБД.

4.2 Ссылочная целостность

Выше мы уже говорили о том, что первичный ключ любой таблицы должен содержать уникальные непустые значения для данной таблицы. Это утверждение является одним из правил *ссылочной целостности* (*referential integrity*). Некоторые (но далеко не все) СУБД могут контролировать уникальность первичных ключей. Ссылочная целостность обеспечивает контроль правильности ссылок и блокирует выполнение ошибочных операций. Например, при попытке присвоить первичному ключу значение, уже имеющееся в другой записи, СУБД сгенерирует диагностическое сообщение, обычно содержащее словосочетание *primary key violation.* Это сообщение в дальнейшем может быть передано в приложение, с помощью которого конечный пользователь манипулирует данными.

Если две таблицы связаны соотношением *master-detail*, внешний ключ *detail-таблицы* должен содержать только те значения, которые уже имеются среди значений первичного ключа *master-таблицы.* Если корректность значений внешних ключей не контролируется СУБД, можно говорить о нарушении ссылочной целостности. В этом случае, если мы удалим из таблицы Customers запись, имеющую хотя бы одну связанную с ней *detail*-запись в таблице Orders, это приведет к тому, что в таблице Orders окажутся записи о заказах, размещенных неизвестно кем. Если же СУБД контролирует корректность значений внешних ключей, то при попытке присвоить внешнему ключу значение, отсутствующее среди значений первичных ключей master-таблицы либо при удалении или модификации записей master-таблицы, приводящих к нарушению ссылочной целостности,

СУБД сгенерирует диагностическое сообщение. Сообщение обычно содержит словосочетание *foreign key violation*, которое в дальнейшем может быть передано в пользовательское приложение.

4.3 Введение в нормализацию данных

Обсудим один из основных принципов проектирования данных — принцип *нормализации.*

Нормализация представляет собой процесс реорганизации данных путем ликвидации повторяющихся групп и иных противоречий в хранении данных с целью приведения таблиц к виду, позволяющему осуществлять непротиворечивое и корректное редактирование данных. Нередко после нормализации к структуре БД добавляется одна или несколько таблиц.

Теория нормализации основана на концепции нормальных форм. Говорят, что таблица находится в данной нормальной форме, если она удовлетворяет определенному набору требований. Теоретически существуют пять нормальных форм, но на практике обычно используются только первые три. Это делается с целью упрощения структуры БД.

4.3.1 Первая нормальная форма

Проиллюстрируем процесс нормализации на примере, использующем данные из базы NorthWind. Предположим, что мы регистрируем все заказанные продукты в следующей таблице:

OrderlD	Product lD	Customerl D	Address	Quantity	OrderDate
10265	17	BLONP	24, place Kleber	30	07.25.96
10265	70	BLONP	24, place Kleber	20	07.25.96
10278	44	BERGS	Berguvsvagen 8	16	08.12.96
10278	59	BERGS	Berguvsvagen 8	15	08.12.96
10278	63	BERGS	Berguvsvagen 8	8	08.12.96
10278	73	BERGS	Berguvsvagen 8	25	08.12.96
10280	24	BERGS	Berguvsvagen 8	12	08.14.96
10280	55	BERGS	Berguvsvagen 8	20	08.14.96
10280	75	BERGS	Berguvsvagen 8	30	08.14.96

OrderlD	Product lD	Customerl D	Address	Quantity	OrderDate
10289	3	BSBEV	Fauntleroy Circus	30	08,26.96
10289	64	BSBEV	Fauntleroy Circus	9	08.26.96
10297	39	BLONP	24, place Kleber	60	09.04.96
10297	72	BLONP	24, place Kleber	20	09.04.96
10308	69	ANATR	Avda. De la Constitucion 22	1	09.18.96
10308	70	ANATR	Avda. De la Constitucion 22	5	09.18.96

Структура этой таблицы имеет следующий вид (рис. 18).

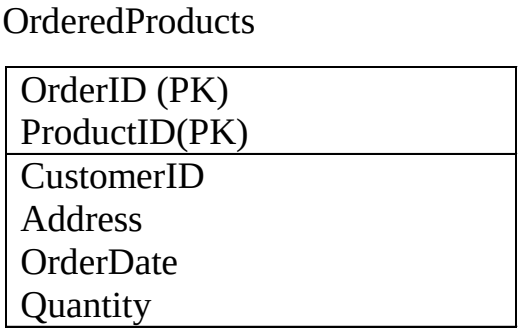


Рис. 18 — Структура ненормализованной таблицы OrderedProducts

Здесь РК обозначает первичный ключ. Чтобы таблица соответствовала *первой нормальной форме*, все значения ее полей должны быть атомарными и все записи — уникальными. Поэтому любая реляционная таблица, в том числе и таблица OrderedProducts, по определению уже находится в первой нормальной форме. Атомарность означает, что в одном поле мы не можем записать все сведения о заказанных продуктах. Одна запись соответствует одному заказанному продукту. Далее, если необходимо, то адрес можно разбить на отдельные поля. В БД содержащих сведения о фамилии имени и отчестве желательно разделить эти сведения на три различных столбца.

Рассмотрим далее. Таблица содержит избыточные данные, например: одни и те же сведения о клиенте повторяются в записи о каждом заказанном продукте. Результатом избыточности данных являются аномалии модификации данных — проблемы, возни-

кающие при добавлении, изменении или удалении записей. Например, при редактировании данных в таблице OrderedProducts могут возникнуть следующие проблемы:

• Адрес конкретного клиента может содержаться в базе данных только тогда, когда клиент заказал хотя бы один продукт.

• При удалении записи о заказанном продукте одновременно удаляются сведения о самом заказе и о клиенте, его разместившем.

• Если, заказчик сменил адрес, придется обновить все записи о заказанных им продуктах.

Некоторые из этих проблем могут быть решены путем приведения базы данных *ко второй нормальной форме.*

4.3.2 Вторая нормальная форма

Говорят, что реляционная таблица находится *во второй нормальной форме,* если она находится в первой нормальной форме и ее неключевые поля *полностью зависят* от всего первичного ключа.

Таблица OrderedProducts находится в первой, но не во второй нормальной форме, так как поля CustomerlD, Address и OrderDate зависят только от поля OrderlD, являющегося частью составного первичного ключа (OrderlD, ProductID). Чтобы перейти от первой нормальной формы ко второй, нужно выполнить следующие шаги:

1. Определить, на какие части можно разбить первичный ключ, так чтобы некоторые из не ключевых полей зависели от одной из этих частей (это *части не обязаны состоять из одной колонки*).

2. Создать новую таблицу для каждой такой части ключа и группы зависящих от нее полей и переместить их в эту таблицу. Часть бывшего первичного ключа станет при этом первичным ключом новой таблицы.

3. Удалить из исходной таблицы поля, перемещенные в другие таблицы, кроме тех их них, которые станут внешними ключами.

Например, для приведения таблицы OrderedProducts ко второй нормальной форме нужно переместить поля CustomerlD,

Address и OrderDate в новую таблицу (назовем ее OrdersInfo), при этом поле OrderID станет первичным ключом новой таблицы (рис. 19).

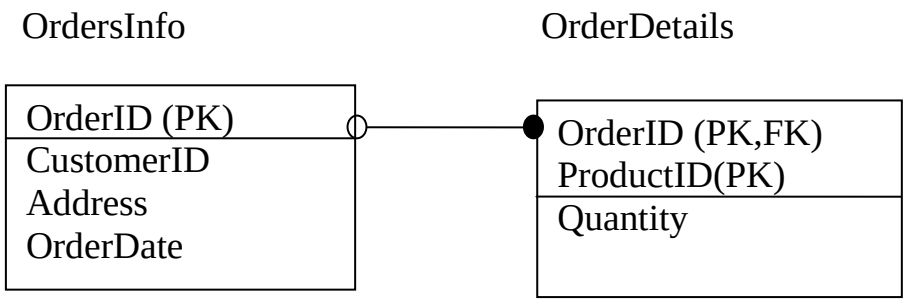


Рис. 19 — Приведение таблицы OrderedProducts ко второй нормальной форме

В результате новые таблицы приобретут вид, показанный ниже.

OrdersInfo			
OrderID	CustomerID	Address	OrderDate
10265	BLONP	24, place Kleber	07.25.96
10278	BERGS	Berguvsvagen 8	08.12.96
10280	BERGS	Berguvsvagen 8	08.14.96
10289	BSBEV	Fauntleroy Circus	08.26.96
10297	BLONP	24, place Kleber	09.04.96
10308	ANATR	Avda. De la Constitucion 22	09.18.96

Однако таблицы, находящиеся во второй нормальной форме, по-прежнему содержат аномалии модификации данных. Например, для таблицы OrdersInfo:

• Адрес конкретного клиента по-прежнему может содержаться в базе данных только тогда, когда клиент заказал хотя бы один продукт.

• Удаление записи о заказе в таблице OrdersInfo приведет к удалению записи о самом клиенте.

• Если заказчик сменил адрес, придется обновить несколько записей (хотя, как правило, их меньше, чем в предыдущем случае). Устранить эти аномалии можно путем перехода к *третьей нормальной форме.*

OrderDatails:		
OrderID	ProductID	Quantity
10265	17	30
10265	70	20
10278	44	16
10278	59	15
10278	63	8
10278	73	25
10280	24	12
10280	55	20
10280	75	30
10289	3	30
10289	64	9
10297	39	60
10297	72	20
10308	69	1
10308	70	5

4.3.3 Третья нормальная форма

Говорят, что реляционная таблица находится в *третьей нормальной форме*, если она находится во второй нормальной форме и все ее не ключевые поля зависят только от первичного ключа.

Таблица OrderDetails уже находится в третьей нормальной форме. Не ключевое поле Quantity полностью зависит от составного первичного ключа (OrderID, ProductID). Однако таблица OrdersInfo в третьей нормальной форме не находится, так как содержит зависимость между не ключевыми полями (она называется *транзитивной зависимостью — transitive dependence* — поле Address зависит от поля CustomerID.

Чтобы перейти от второй нормальной формы к третьей необходимо выполнить следующие шаги:

1. Определить все поля (или группы полей), от которых зависят другие поля.

2. Создать новую таблицу для каждого такого поля (или группы полей) и группы зависящих от него полей и переместить их в эту таблицу. Поле (или группа полей), от которого зависят

все остальные перемещенные поля, станет при этом первичным ключом новой таблицы.

3. Удалить перемещенные поля из исходной таблицы, оставив лишь те из них, которые станут внешними ключами.

Для приведения таблицы Orderslnfo к третьей нормальной форме создадим новую таблицу Customers и переместим в нее поля CustomerID и Address. Поле Address из исходной таблицы удалим, а поле CustomerID оставим — теперь это внешний ключ (рис. 20):

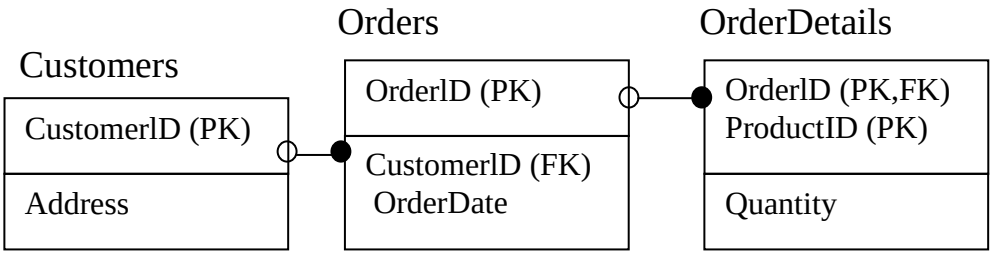


Рис. 20 — Приведение таблицы Orderslnfo к третьей нормальной форме

Orders:		
OrderID	CustomerID	OrderDate
10265	BLONP	07.25.96
10278	BERGS	08.12.96
10280	BERGS	08.14.96
10289	BSBEV	08.26.96
10297	BLONP	09.14.96
10308	ANATR	09.18.96

Customers:	
CustomerID	Address
ANATR	Avda. De la Constitucion 2222
BERGS	Berguvsvagen 8
BLONP	24, place Kleber
BSBEV	Fauntleroy Circus

Итак, после приведения исходной таблицы к третьей нормальной форме таблиц стало три — Customers, Orders и OrderDetails. Например, после приведения рассмотренной выше базы данных к третьей нормальной форме налицо следующие улучшения:

• Сведения об адресе клиента можно хранить в базе данных, даже если это только потенциальный клиент, еще не разместивший ни одного заказа.

• Сведения о заказанном продукте можно удалять, не опасаясь удаления данных о клиенте и заказе.

Изменение адреса клиента или даты регистрации заказа теперь требует изменения только одной записи.

Если в отношении, находящемся в третьей нормальной форме, отсутствуют многозначные зависимости, но имеются другие зависимости от ключа, то третья нормальная форма будет иметь аномалии операций. В этом случае рассматривают усиленную третью нормальную форму (форму Бойса–Кодда).

4.3.4 Нормальная форма Бойса–Кодда

Отношение находится в нормальной форме Бойса–Кодда, если оно находится в третьей нормальной форме и каждый детерминант отношения является возможным ключом отношения. Атрибут (или комбинацию атрибутов), от которого какой-либо другой атрибут зависит функционально (полно), называют детерминантом. Рассмотрим пример. Пусть отношение, которое моделирует сдачу текущей сессии, имеет следующую структуру:

(Номер зачетной книжки, Идентификатор студента, Дисциплина, Дата, Оценка)

Это отношение находится в третьей нормальной форме. Но в данном отношении у нас есть два детерминанта — Номер зачетной книжки и Идентификатор студента. Известно, что каждому студенту ставится в соответствие один номер зачетной книжки и один идентификатор. Для приведения к нормальной форме Бойса–Кодда надо разделить отношение, например на два со следующими схемами:

(Идентификатор студента, Дисциплина, Дата, Оценка)
(Номер зачетной книжки, Идентификатор студента)

или

(Номер зачетной книжки, Дисциплина, Дата, Оценка)
(Номер зачетной книжки, Идентификатор студента)

4.3.5 Четвертая нормальная форма

Четвертая нормальная форма запрещает хранить независимые компоненты в одной таблице, когда между этими компонентами существуют отношения многие-ко-многим. Можно решить проблему приведения к четвертой нормальной форме, поместив каждый многозначный атрибут в отдельную таблицу вместе с ключом от которого этот атрибут зависит.

Допустим, существует отношение, моделирующее сдачу экзаменов:

(Номер зачетной книжки, Группа, Дисциплина)

Перечень дисциплин, которые должен сдавать студент в сессию зависит не от его Фамилии, а от группы, в которой учится данный студент. Группа зависит от учебного плана, в котором перечислены все дисциплины, изучаемые по данной специальности.

В приведенном выше отношении существуют две многозначные зависимости:

Группа — Дисциплина
Группа — номер зачетки

Группа определяет список студентов, которые в ней учатся. Приведенное выше отношение, для соответствия четвертой нормальной форме декомпозируют на два: (номер зачетной книжки, Группа), (Группа, Дисциплина). Операции модификации теперь упрощаются: добавление нового студента связано с добавлением одного кортежа в первое отношение, а добавление дисциплины связано с добавлением одного кортежа во второе отношение. Кроме того, можем хранить любое количество групп, с определенным перечнем дисциплин, в которые еще не зачислены студенты.

4.3.6 Пятая нормальная форма

Пятая нормальная форма была предложена для того, чтобы исключить аномалии, связанные с особым типом ограничительных условий, называемых совместимыми зависимостями. Эти зависимости имеют в основном теоретический интерес. Не было

предложено правил приведения к пятой нормальной форме. На практике не находит применения.

Нормализация устраняет избыточность данных, что позволяет снизить объем хранимых данных и избавиться от описанных выше аномалий их изменения.

4.4 Как проектируют базы данных

Обычно современные СУБД содержат средства, позволяющие создавать таблицы и ключи. Существуют и утилиты, поставляемые отдельно от СУБД (и даже обслуживающие несколько различных СУБД одновременно), позволяющие создавать таблицы, ключи и связи.

Еще один способ создать таблицы, ключи и связи в базе данных — это написание так называемого DDL-сценария (DDL — Data Definition Language).

Наконец, есть еще один способ, который становится все более и более популярным, — это использование специальных средств, называемых CASE-средствами (CASE означает Computer-Aided System Engineering). Существует несколько типов CASE-средств, но для создания баз данных чаще всего используются инструменты для создания диаграмм «сущность-связь» (entity-relationship diagrams, E/R diagrams). С помощью этих инструментов создается так называемая *логическая* модель данных, описывающая факты и объекты, подлежащие регистрации в ней (в таких моделях прототипы таблиц называются сущностями (entities), а поля — их атрибутами (attributes). После установления связей между сущностями, определения атрибутов и проведения нормализации создается так называемая *физическая* модель данных для конкретной СУБД, в которой определяются все таблицы, поля и другие объекты базы данных. После этого можно сгенерировать либо саму базу данных, либо DDL-сценарий для ее создания.

5. ЯЗЫК ФОРМИРОВАНИЯ ЗАПРОСОВ К БАЗЕ ДАНЫХ

Structured Query Languge (SQL)

Прототип языка SQL был разработан в конце 1970-х годов в компании IBM (SQL) в первом прототипе реляционной СУБД System R. Первый международный стандарт языка был принят в 1989 г. Бурное развитие информационных технологий и СУБД потребовало расширение стандарта. И в 1992, а затем 1999 г. были приняты соответственно второй и третий стандарты языка SQL. Современные перспективные СУБД содержат в своем составе SQL, соответствующий одному из стандартов. SQL/3 является полным языком и содержит не только операторы запросов, но и язык описания, манипулирования данными и операторы, предназначенные для администрирования БД.

Ниже перечислены основные операторы SQL.

Операторы определения данных:

CREATE TABLE — создать таблицу;

DROP TABLE — удалить таблицу;

ALTER TABLE — изменить таблицу;

CREATE VIEW — создать представление;

ALTER VIEW — изменить представление;

DROP VIEW — удалить представление;

CREATE INDEX — создать индекс;

DROP INDEX — удалить индекс

Операторы манипулирования данными:

DELETE — удалить строки;

INSERT — вставить строку;

UPDATE — обновить строку.

Язык запросов:

SELECT — выбрать строки.

Средства управления транзакциями:

COMMIT — завершить транзакцию;

ROLLBACK — откатить транзакцию;

SAVEPOINT — сохранить промежуточную точку выполнения транзакции.

Средства администрирования данных:

ALTER DATABASE— изменить БД;

ALTER DBAREA — изменить область хранения БД;
ALTER PASSWORD — изменить пароль;
CREATE DATABASE — создать БД;
CREATE DBAREA — создать область хранения;
DROP DATABASE — удалить БД;
DROP DBAREA — удалить область хранения БД;
GRANT — предоставить права;
REVOKE — лишить прав.
Программный SQL:
DECLARE — определить курсор для запроса;
OPEN — открыть курсор;
FETCH — считать строку из множества строк, определенных курсором;
CLOSE — закрыть курсор;
PREPARE — подготовить оператор SQL к динамическому выполнению;
EXECUTE — выполнить оператор SQL, ранее подготовленный к динамическому выполнению.

Необходимо отметить, что в каждой СУБД своя специфика реализации языка. Это, как правило, касается типов данных, способов задания констант, арифметических и логических операций и т.д. Здесь рассмотрим операторы манипулирования данными и оператор SELECT.

5.1 Оператор выбора

Рассмотрим один из наиболее сложных операторов языка — оператор SELECT. Его синтаксис приведен ниже.

SELECT
[ALL | DISTINCT]
[<псевдоним>.]<выражение>[AS<колонка>]
[,[<псевдоним>.] **<список выбора>**[AS<колонка>]...]
FROM <БД>[,[<БД>...]
[WHERE <условие связи>
[AND <условие связи>...]
[AND/OR <условие отбора>
[AND/OR <условие отбора>...]]]

[GPOUP BY <колонка>[,<колонка>...]]
[ORDER BY <колонка>[,<колонка>...]]]
[HAVING <условие отбора>]

Здесь ключевое слово ALL означает, что в результирующую выборку будут включаться все записи, удовлетворяющие запросу. DISTINCT предотвращает повторный вывод одних и тех же записей.

<выражение> может содержать:

– поле базы данных;

– константу; указанное значение константы появится в каждом ряду результата запроса;

– выражение, которое может включать пользовательскую функцию;

– параметр FROM задается всегда. Определяет имена всех баз данных, которые принимают участие в запросе;

– <выражение> может быть полем, имя может быть составным т.е. из нескольких констант.

Необязательная фраза AS <колонка> специфицирует заголовок колонки в выводе результата запроса. Это полезно, когда имеются <выражения> или содержатся функции работы с полями и требуется присвоить колонке в результате соответствующее имя.

<колонка> может быть выражением.

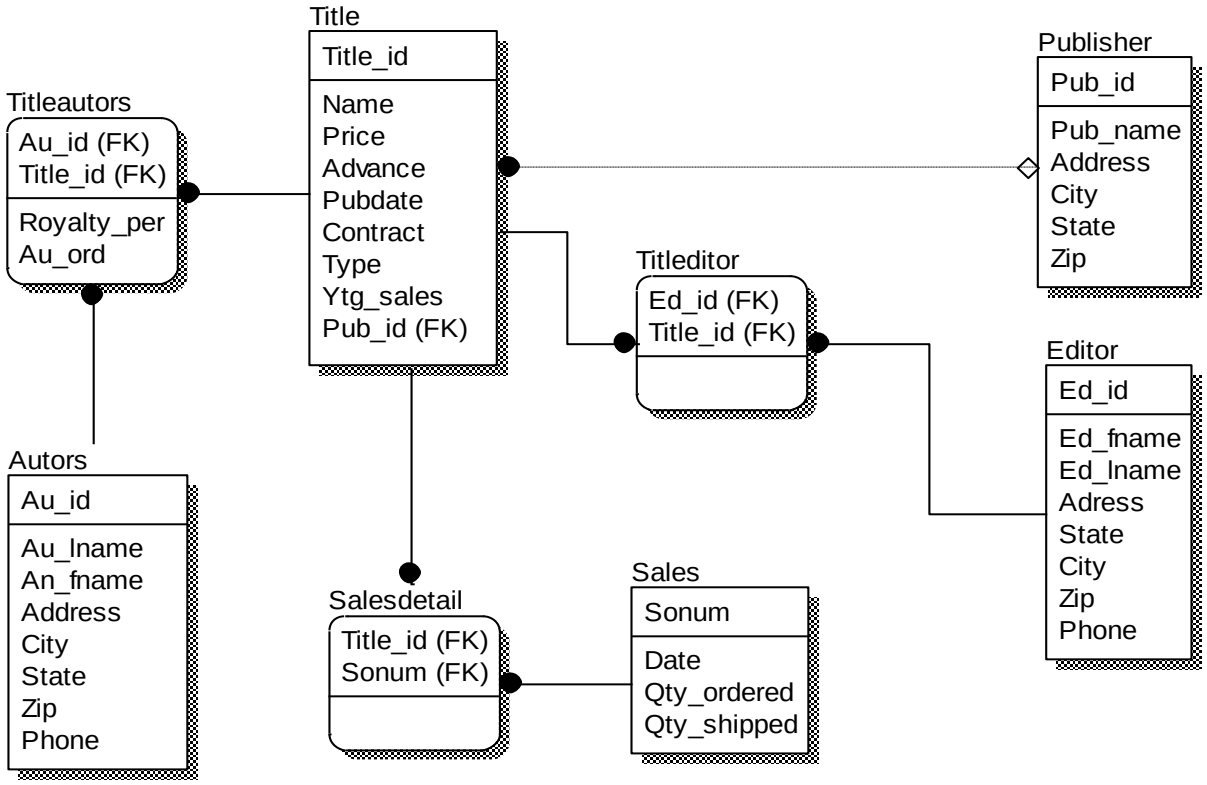
Если в разных базах одни и те же имена и они одновременно выводятся на экран, то колонки получат одинаковое название и букву алфавита fam; fam a, fam b...

Если не устраивают имена колонок данных по умолчанию, можно их обозначить по-своему, используя выражение:

[AS <Новое наименование колонки>]

Псевдоним БД, откуда берется нужное поле может, быть назначен в команде SELECT. Вне команды действия не имеет. Рассмотрим примеры. Будем использовать спроектированную учебную базу BOOK, структура которой приведена ниже.

База спроектирована для компании, занимающейся выпуском книг и содержащей несколько издательств.



Руководство компанией интересует информация о книгах, выпускаемых в издательствах, об авторах книг и редакторах, а также о финансовом положении компании. При этом в базе данных должно было быть учтено, что:

– автор может написать несколько книг;

– книга может быть написана несколькими авторами;

– порядок следования авторов в заглавии книги влияет на гонорар;

– редактор может работать с несколькими книгами и несколько редакторов с одной;

– в заказе на покупку может быть перечислено несколько книг.

Исходя из вышесказанного, были выделены объекты и определены их характеристики.

Объект Autors — авторы, книги которых опубликованы компанией, имеют следующие атрибуты: имя автора, полный адрес, где он проживает, телефон.

Объект Titles — книги, имеют атрибуты: название книги, ее стоимость, дату выпуска.

Объект Editors — редакторы, работающие на компанию, имеют атрибуты аналогичные объекту авторы: имя, адрес, телефон.

Объект Publishers — имеет атрибуты: наименование издательства, которыми владеет компания, его адрес, и телефон. В результате проектирования получим совокупность таблиц приведенную выше.

Допустим, необходимо вывести значения всех столбцов. Оператор SELECT в этом случае запишется:

```
SELECT *
FROM  PUBLISERS
```

Pub_id	Pub_name	Address	City	State	Zip

Тот же самый результат можно получить, перечислив в списке выбора все поля таблицы.

```
SELECT PUB_ID, PUB_NAME, ADDRESS, CITY, STATE, ZIP
FROM PUBLISHERS
```

Если необходимо осуществить выбор отдельных столбцов, то в списке выбора перечисляют необходимые столбцы в порядке их отображения на экране.

```
SELECT   PUB_ID, PUB_NAME
FROM PUBLISHERS
```

С помощью оператора возможно переименование столбцов и задание имен выражениям. При выводе результатов запроса каждый столбец по умолчанию получает заголовок, совпадающий с его именем в БД. Поскольку столбцы в БД обычно имеют сокращенные имена, желательно им присвоить полные. Выполняется это с помощью предложения AS. В списке выбора пишут:

имя столбца AS имя заголовка

```
SELECT   PUB_NAME  AS   PUBLISHERS, PUB_ID
FROM PUBLISHERS
```

Таблица с поименованными столбцами приведена ниже.

Publishers	pub_id

В большинстве систем ширина отображаемых столбцов устанавливается по максимальной длине заголовка (но не уже находящихся в них данных). Как правило, в заголовках столбцов нельзя использовать кавычки и пробелы, поэтому нет возможности применять русские названия столбцов. Если в списке выбора задано выражение, то для него тоже можно создать заголовок.

Например, если мы хотим посмотреть, как изменится стоимость книг, если мы увеличим их стоимость на 10%, то запрос может выглядеть так:

```
SELECT TITLE, PRICE AS OLD_PRICE, PRICE * 1.1 AS NEW_PRICE
FROM  TITLE
```

TITLE	OLD_PRICE	NEW_PRICE

Пояснения в строках

Например:

```
SELECT 'СТОИМОСТЬ КНИГИ', TITLES, 'СТАЛА',
PRICE*1.1 AS  NEW_PRICE
FROM  TITLE
```

В списках с числовыми данными и константами можно выполнять следующие арифметические действия: +, –, *, /. (в некоторых системах введена операция остаток от деления — %)

Таблицам можно присвоить имена-псевдонимы, что бывает полезно для осуществления соединения таблицы с самой собой или для доступа из вложенного подзапроса к текущей записи внешнего запроса.

Например:

```
SELECT P. PUB_ID, P.PUB_NAME
FROM  PUBLISHERS P
```

p — здесь псевдоним, p.pub_id — полное наименование поля.

Выполнение запроса

```
SELECT * FROM AUTORS, TITLES
```

Соответствует декартову произведению таблиц AUTORS, TITLES.

5.1.1 Определение критерия отбора данных

Отбор данных в выборке производится посредством ключевого слова WHERE:

WHERE <условие связи> [AND <условие связи>...]

[ARD/OR <условие отбора> [AND/OR < >...]]

<условие связи> применяется, если выборка делается более чем из одной базы, и указывает критерий, которому должны удовлетворять поля из разных баз. Здесь используются операции отношений =, #, >, <, >=, <=, допускается применение нескольких критериев, соединенных знаком AND.

Если включить в запрос две базы данных и не специфицировать условие связи, то каждая запись из первой базы данных будет соединяться с каждой записью из второй базы данных при выполнении условия фильтрации.

<условие отбора> специфицирует критерий, которому должны удовлетворять записи, чтобы попасть в результат действия запроса. Условия фильтрации можно соединять операторами AND и OR. Можно также использовать оператор NOT для реверса значения логического выражения.

Операторы условия =, >, <, >=, <=, !=, < > применяются к данным имеющим тип: числовой, символьный, дата.

Например, необходимо найти все книги стоимостью более 250 рублей.

```
SELECT  TITLE,    PRICE
FROM     TITLE
WHERE   PRICE  > 250
```

Запрос, в котором необходимо найти книги, выпущенные издательством до 1999г., выглядит следующим образом:

```
SELECT    TITLE,   TITLE_ID, PRICE
FROM     TITLE
WHERE  PUB_DATE < {01. 01. 1999}
```

Допустим, необходимо найти авторов живущих не в Томске и их телефоны:

```
SELECT  AU_ID, PHONE
FROM  AUTORS
WHERE  CITY != 'ТОМСК'
```

Возможносовместное использование условных и логических операторов

Пусть нужно найти книги по Windows стоимостью выше 250 руб. и затратами на выпуск ниже 2 000 000.

```
SELECT   TITLE, TYPE, PRICE, ADVANCE
FROM  TITLE
WHERE TYPE = 'WINDOWS'   AND PRICE > 250    AND
ADVANCE < 2000000
```

При составлении логических выражений нужно быть внимательным. Рассмотрим два примера. В первом примере в выборку попадут все книги по бизнесу независимо от затрат на их производство и книги по искусству, затраты на выпуск которых превысили сумму в 5 500 000 рублей.

```
SELECT   TITLE_ID, TYPE, PRICE, ADVANCE
FROM TITLE
WHERE  TYPE ‘БИЗНЕС’
 OR TYPE = ‘ИСКУССТВО’
 AND ADVANCE > 5 500 000
```

Во втором случае за счет расставленных скобок в выборку попадут все книги по искусству и бизнесу затраты на выпуск которых превысили 5500000.

```
SELECT   TITLE_ID, TYPE, ADVANCE
FROM TITLE
WHERE (TYPE = 'БИЗНЕС '  OR TYPE = 'ИСКУССТВО')
AND ADVANCE > 5 500 000
```

Еще один пример. Необходимо найти книги, затраты на выпуск которых не окупились.

Доход от продажи определяется как ytd_sales * price, advance — сумма затрат. Книги выпущены два года назад.

```
SELECT          TITLE_ID, TYRE, PRICE, ADVANCE,
YTD_SALES
FROM TITLE
WHERE PRICE * YTD_SALES < ADVANCE
AND PUB_DATE < {10.1. 2003}
```

Определение диапазона реализации можно проводить с помощью знаков > <,а также с помощью ключа BETWEEN (NOT BETWEEN), который проверяет, находится ли (или нет) выражение в заданном диапазоне.

<выражение> BETWEEN <нижнее значение> AND <верхнее значение>

Допустим, необходимо найти книги, количество проданных экземпляров у которых между 4095 и 12000 включительно.

```
SELECT   TITLE_ID, YTD_SALES
FROM  TITLE
WHERE YTD_SALES BETWEEN 4095 AND 12000
```

Запрос, в котором число проданных экземпляров меньше 4095 или больше чем 12000, запишется как

```
SELECT   TITLE_ID, YTD_SALES
FROM  TITLE
WHERE YTD_SALES NOT BETWEEN 4095 AND 1200
```

То же самое получается при следующем запросе:

```
SELECT   TITLE_ID, YTD_SALES
FROM  TITLE
WHERE YTD_SALES <  4095 OR  YTD_SALES
```

Списки IN, not IN позволяют выбрать значения из заданного списка.

<выражение> IN (<выражение>, <выражение>,...)

Выбрать авторов, проживающих в г.г. Томск и Сургут.

```
SELECT   A_LNAME, STSTE
FROM AUTORS
WHERE CYTI IN ('Томск', 'Сургут')
```

Найти имена авторов, которые получают меньше 50% от суммарного гонорара за книги, написанные в соавторстве. Здесь приведен пример запроса с подзапросом.

```
SELECT   AU_LNAME, AU_FNAME
FROM AUTORS
WHERE AU_ID  IN
 (SELECT  AU_ID FROM TITLEAUTORS
 WHERE ROYALTY_PER < 0.5)
```

В предложении WHERE для поиска нужных подстрок удобно применять шаблоны.

<выражение> LIKE <шаблон>

В качестве части этого выражения можно использовать специальные символы % (процент) и _<(подчеркивание)>. Подчеркивание _ означает подстановку одного любого символа, а процент % — произвольную последовательность таких символов.

Например, выбрать книги, в названии которых есть слово «компьютер».

```
SELECT TITLE_ID, NAME, PRICE
FROM TITLE
WHERE NAME LIKE  '%компьютер%'
```

5.1.2 Сортировка результатов запроса

Сортировка результатов запроса производится с использованием ключевого слова ORDER BY.

ORDER BY <колонка> [ASC/DESC][,<колонка>,...] — задает упорядочение по колонке, колонкам.

Фраза DESC вызывает упорядочение по убыванию значения соответствующей <колонки>. Фраза ASC специфицирует упорядочение по возрастанию и принимается по умолчанию.

Пример сортировки по цене (список цен идентификационных номеров издателей):

```
SELECT PRICE, TITLE_ID, PUB_ID
FROM TITLE
ORDER BY PRICE
```

Хорошо бы книги в каждой ценовой категории, выпущенные одним издательством, в списке были рядом. Для этого в список ORDER BY добавим столбец pub_id.

```
SELECT PRICE, TITLE_ID, PUB_ID
FROM TITLE
ORDER BY PRICE,  PUB_ID
```

Количество уровней сортировки может быть любым. Порядок сортировки задается в предложении ORDER BY. Порядок следования столбцов при выводе задается списком выбора. В операторе SELECT список выбора может не совпадать со списком сортировки.

Ключевые слова ASC, DESC определяют направление сортировки: по возрастанию, по убыванию соответственно и действует только на один столбец.

В примере упорядочим идентификационные номера по цене и затем по издательствам, причем по цене упорядочивание проводится по убыванию, а по издательствам по возрастанию.

```
SELECT PUB_ID, PRICE, TITLE_ID,
FROM TITLE
ORDER BY PRICE DESC,  PUB_ID
```

При выводе возможна сортировка выражений. Допустим, необходимо результаты отсортировать по издателям, затем по продажам. В этом случае в предложении ORDER BY ставится номер позиции (натуральное число без знака), соответствующий выражению в списке выбора. В результате запрос будет выглядеть следующим образом:

```
SELECT  PUB_ID, PRICE * YTD_SALES, PRICE, TITLE_ID
FROM TITLE
ORDER BY PUB_ID, 2
```

pub_id		price	title_id

В список сортировки можно добавить предложение DESC, тогда поле продаж будет отсортировано по убыванию.

5.1.3 Агрегирующие функции

Агрегирующая функция рассматривает множество строк таблицы и выдает только одно значение. Приведем список агрегирующих функций.

SUM([DISTINCT] выражение) — сумма (различных) выражений.

AVG([DISTINCT] выражение) — средняя величина (различных) значений.

COUNT ([DISTINCT] выражение) — количество (различных) ненулевых значений.

COUNT (*) — полное количество выбранных строк, включая пустые строки и дубликаты.

MAX (выражение) — максимальное значение.

MIN (выражение) — минимальные значение.

При использовании агрегирующих функций необходимо учитывать следующее:

– их можно записывать в списке выбора или в предложении HAWING;

– нельзя применять в предложении WHERE;

– перед вычислением значения функции все неопределенные значения аргумента исключаются;

– если определено предложение DISTINCT, то повторяющиеся значения аргумента в подсчете значения функции не участвуют;

– если аргумент пуст, т.е. содержит неопределенные значения, то функция COUNT всегда возвращает значение 0, а другие агрегирующие функции — неопределенное значение;

– функции SUM и AVG работают только с аргументами числовых типов;

– функции COUNT, MAX, MIN могут использоваться с аргументами любых типов.

Приведем несколько примеров использования функций.

1. SELECT YTD_SALES
   FROM TITLE
2. SELECT SUM (YTD_SALES)
   FROM TITLE

В первом случае соответствующие значения выдаются виде таблицы, т.е. выдается столько строк, сколько их есть в таблице TITLE. Если использовать агрегирующую функцию SUM, то в результате получим общую сумму, и это будет одно значение.

Одно выдаваемое значение можно пояснить, например, при записи следующего запроса:

SELECT 'ИТОГ ', SUM (YTD_SALES)
FROM TITLE

на экране можно увидеть:

Итог: 97445

Приведем примеры использования функции COUNT.

Допустим, необходимо узнать количество авторов, сотрудничающих с издательством. Запрос будет выглядеть так:

SELECT COUNT (AU_FNAME)
FROM AUTORS

В том случае, когда необходимо узнать количество разных фамилий авторов, сотрудничающих с издательством, запрос запишется с использованием предложения DISTINCT:

SELECT COUNT (DISTINCT AU_FNAME)
FROM AUTORS

5.1.4 Группировка данных и построение отчетов

В менеджменте часто требуется статистическая информация о каждой группе в множестве групп. Для этого используется предложение GROUP BY, которое разделяет таблицу на наборы. GROUP BY неразрывно связано с агрегирующими функциями и предложением HAVING.

Пример.

```
SELECT AVG (PRICE)
FROM TITLE

SELECT  TYPE, AVG (PRICE)
FROM TITLE
GROUP BY TYPE
```

В результате первого запроса получим среднюю стоимость книг. Это будет единственное значение. Во втором запросе книги группируются по типу. Строки помещаются в одну группу тогда и только тогда, когда у них совпадает поле TYPE. Затем фраза SELECT применяется к каждой группе, т.е. внутри каждой группы считается средняя цена. Для каждой группы SELECT выводит TYPE и среднее значение по типу.

TYPE	
Спорт	86.78
Домашнее хозяйство	190.43

Как правило, в большинстве реализаций SQL элемент из списка GROUP BY должен присутствовать в списке выбора SELECT.

Путем сортировки одновременно по нескольким элементам можно создавать группы внутри других групп.

Пусть необходимо подсчитать количество книг по каждому типу, выпущенному каждым издательством. Здесь группировка производится сначала по издательству, затем по типу.

```
SELECT PUB_ID, TYPE, COUNT (TYPE)
FROM TITLE
GROUP BY PUB_ID, TYPE
```

Результат:

Pub_id	type	
0732	спорт	5
0732	бизнес	4
0877	бизнес	2
0877	спорт	6
0877	искусство	4

Если выполнить предложение GROUP BY без агрегирующих функций, то оно будет напоминать предложение DISTINCT, т.е. разделять таблицу на группы и из группы брать одно значение. Фактически предложение GROUP BY и агрегирующие функции были созданы друг для друга.

Допустим, необходимо найти средние затраты и сумму доходов от продаж по каждому типу книг:

```
SELECT TYRE, AVG(ADVANCE), SUM(YTD_SALES*PRICE)
FROM TITLE
GROUP BY TYPE
```

TYPE		
Бизнес	62.81	307587
Компьютеры	75.000	242784
Спорт	42.00	99398

GROUP BY возможно применять вместе с предложением WHERE. В этом случае сначала находятся все строки, удовлетворяющие предложению WHERE, затем оставшиеся строки группируются в соответствии с предложением GROUP BY. GROUP BY разделяет строки на наборы, но при этом не упорядочивает их. Для упорядочивания результатов нужно использовать предложение ORDER BY.

Например. Найти среднюю стоимость книг по каждому типу, затраты на которые превысили 50000, и упорядочить результаты по цене. Запрос выглядит следующим образом:

```
SELECT TYPE, AVG (PRICE)
FROM TITLE
WHERE ADVANCE > 50000
GROUP BY TYPE
ORDER BY 2
```

TYPE	
Бизнес	2.9.9
Спорт	30.1
Анатомия	42.3

Условие, накладываемое на группировки, задается предложением HAVING. Предложение HAVING похоже на предложение WHERE, но HAVING работает не с отдельными записями таблицы, а с группами. Последовательность действий следующая. Сначала для всего запроса выполняется предложение WHERE, затем GROUP BY, SELECT и уже к сгруппированным данным применяется условие, записанное в предложении HAVING.

Пусть необходимо группировать данные по типу выпускаемых книг и при этом исключить из рассмотрения наборы, содержащие только одну книгу.

```
SELECT TYPE, COUNT(*)
FROM TITLE
GROUP BY TYPE
HAVING COUNT(*) > 1
```

Условия в предложении HAWING могут объединяться с помощью операторов AND, OR, NOT.

Приведем в качестве примера запрос, содержащий выражения WHERE, GROUP BY, ORDER BY, HAWING. Пусть необходимо сгруппировать строки из таблицы TITLES по издателям, при этом включить в конечный результат только группы издателей с идентификационными номерами большими 0800, суммарными затратами большими 750 000 рублей, средней ценой книг меньше 300 рублей и без учета книг стоимостью меньше 50 рублей.

```
SELECT PUB_ID, SUM(ADVANCE), AVG(PRICE)
FROM TITLE
WHERE PRICE >= 50
GROUP BY PUB_ID
HAVING SUM(ADVANCE) > 750000
AND AVG(PRICE) < 300
AND PUB_ID > ‘0800’
ORDER BY PUB_ID
```

Использование нулевых значений

При рассмотрении основных предложений оператора SELECT не рассматривался вопрос, связанный с использованием нулевых значений.

В том случае, когда информация неполна, неизвестна на настоящий момент времени, приходится иметь дело с так называемыми нулевыми (NULL) значениями. Нулевые значения появляются в том случае, если пользователь вводит данные и не знает, какую информацию нужно вводить в некоторых полях. В этом случае система автоматически вводит нулевые значения. При появлении нулевых значений не действуют стандартные правила сравнения: одно неопределенное значение не равно другому неопределенному значению. Нулевые значения приводят к появлению трехзначной логики в логических выражениях.

Таблица истинностей для трехзначной логики приведена ниже.

a	b	not a	a and b	a or b
1	1	0	1	1
1	0	0	0	1
1	–	0	–	1
0	1	1	0	1
0	0	1	0	0
0	–	1	0	–
–	1	–	–	1
–	0	–	0	–
–	–	–	–	–

Здесь неопределенное значение показано прочерком (**–**).

При группировке все нулевые значения, как правило, помещаются в одну группу. При вычислениях строки с нулевым значением столбца можно извлекать из таблицы с помощью специального условия IS [NOT] NULL. Например, чтобы найти все книги с ненулевыми затратами, можно использовать запрос:

```
SELECT TITLE_ID, ADVANCE
FROM TITLE
WHERE ADVANCE IS NOT NULL
```

5.1.5 Объединение таблиц и сложный анализ данных

В большинстве систем объединение таблиц осуществляется в предложении WHERE оператора SELECT. В одном операторе SELECT может объединяться несколько таблиц. Объединение, как правило, проводится по ключевым столбцам,

Например. Необходимо узнать имена редакторов книги с идентификационным номером «CW97».

```
SELECT ED_LNAME, ED_FNAME
FROM EDITOR, TITLEDITOR
WHERE EDITOR. ED_ID = TITLEDITOR.ED_ID
   AND TITLEDITOR.TITLE_ID = "SW97"
```

Поскольку две таблицы содержат столбцы с одинаковыми именами, то для того чтобы их можно было различить, помещают перед именем столбца имя исходной таблицы, отделяя его точкой.

Рассмотрим этапы обработки системой данного запроса. Сначала обрабатывается фраза FROM. Однако в этом случае, поскольку в команде заданы две таблицы, система создает декартово произведение строк этих таблиц. Если в предложении FROM указано более двух таблиц, то создается декартово произведение всех таблиц, перечисленных в команде. После создания гигантской реляционной таблицы система выполняет предложение WHERE. Каждая строка таблицы, созданной предложением FROM, проверяется на выполнение условия WHERE.

В нашем случае фраза WHERE содержит два условия:

```
EDITOR. ED_ID = TITLEDITOR.ED_ID
TITLEDITOR.TITLE_ID = "SW97"
```

Первое из этих условий — условие соединения, которое означает, что в любой выбранной строке значение ED_ID из таблицы EDITOR должно совпадать со значением ED_ID из таблицы TITLEEDITOR. Все строки, не удовлетворяющие равенству, исключаются из таблицы произведения. Далее рассматривается второе условие.

При проектировании структуры базы данных в таблицах нужно предусматривать столбцы, по которым в дальнейшем будет производиться объединение таблиц. Чаще всего для этой цели служат первичные и соответствующие им внешние ключи.

В запросе могут использоваться первичные ключи из двух разных таблиц, как в приведенном ниже примере.

```
SELECT ED_LNAME
FROM EDITOR, AUTORS
WHERE ED_ID = AN_ID
```

Нашли авторов, которые когда-либо выполняли обязанности редакторов.

Рассмотрим запрос

```
SELECT *
FROM EDITORS, PUBLISHERS
WHERE EDITOR.CITY=PUBLISHER.CITY
AND PUB_NAME='ACT'
```

В данном запросе необходимо найти редакторов живущих в городе, где расположено издательство 'ACT'. В списке выбора указана *. В этом случае будут выведены все столбцы из обеих таблиц, причем порядок следования столбцов соответствует порядку следования их в таблице, а порядок следования таблиц перечислен в предложении FROM.

Псевдонимы (алиасы) улучшают читаемость запросов. Необходимо получить информацию об авторах всех книг типа «Искусство».

```
SELECT AN_LNAME, AN_FNAME, TITLE
FROM AUTOR A, TITLES T, TATLEAUTORS TA
WHERE A.AU_ID = TA.AU_ID
AND TA.TITLE_ID = T.TITLE_ID
AND T.TYPE = 'ИСКУССТВО'
```

Можно производить самообъединение двух таблиц. В этом случае создаются две копии таблицы с разными псевдонимами. Затем копии соединяются с использованием предложения WHERE.

Например, надо найти авторов, имеющих один и тот же индекс и живущих в городе Новосибирске. Чтобы в результирующей таблице не было одинаковых строк, применим слово DISTINCT, а авторы не были объединены сами с собой, в предложение WHERE добавим условие AU1.AU_ID != AU2.AU_ID.

```
SELECT DISTINCT AU1.AU_FNAME, AU1.AU_LNANE,
AU1.ZIP
FROM AUTORS AU1, AUTORS AU2
```

```
WHERE AU1.ZIP=AU2.ZIP AND AU1.CITY = ‘Новосибирск’
AND AU1.AU_ID != AU2.AU_ID
```

Были рассмотрены объединения, основывающиеся на равенстве или совпадении столбцов. Помимо равенства, могут использоваться следующие логические операторы: >, >=, <, <=, != или <>. Объединения, построенные только на операторах сравнения, называются тета-объединениями.

Внешние объединения осуществляются при помощи операторов =* и *=. Во внешнее объединение включаются строки одной из таблиц не удовлетворяющие условиям объединения.

*= — позволяет включить все строки первой таблицы;

=* — позволяет включить все строки второй таблицы.

Как и в случае обычных объединений, для ограничения результатов внешнего объединения можно использовать условные операторы.

5.1.6 Подзапросы

Операторы SQL можно вкладывать друг в друга. Подзапрос — это оператор SELECT, вложенный в предложение WHERE, HAVING или SELECT другого оператора SELECT, в оператор INSERT, UPDATE или DELETE, в другой подзапрос.

Рассмотрим синтаксис подзапроса.

```
SELECT [DISTINCT]
FROM список таблиц
WHERE
{выражение {[NOT] in | оператор сравнения [ANY | ALL]} |
[NOT] EXISTS}
(SELECT [DISTINCT] список выбора подзапроса
FROM  список таблиц
WHERE  условия)
[GROUP BY  список группировки
[HAVING условие]]
[ORDER BY порядок ]
```

Подзапрос заключен в круглые скобки. Подзапросы имеют две формы: коррелированную и некоррелированную.

Найти названия издательств, выпускающих книги по бизнесу.

```
FROM PUBLISHERS
  WHERE PUB_ID IN
        (SELECT PUB_ID
        FROM TITLE
        WHERE TYPE = ‘БИЗНЕС’)
```

Внутренний запрос выполняется независимо, передавая результаты во внешний запрос.

```
SELECT PUB_NAME   (коррелированный запрос)
  FROM PUBLISHER P
  WHERE 'БИЗНЕС’ IN
        (SELECT TYPE
        FROM TITLE
        WHERE PUB_ID = P.PUB_ID)
```

Внутренний запрос для своего выполнения должен получить данные из внешнего запроса.

Можно этот подзапрос сформулировать как запрос на объединение:

```
SELECT DISTINCT PUB_NAME
FROM PUBLISHER P, TITLES T
WHERE P.PUB_ID = T.PUB_ID
AND TYPE = ‘БИЗНЕС’
```

Некоррелированный подзапрос реализуется за два шага. Сначала внутренний запрос возвращает идентификационные номера издательств, которые опубликовали книги по бизнесу, затем эти сведения передаются во внешний запрос, который отыскивает названия издательств.

Принадлежность столбцов задается в предложениях FROM.

Коррелированная обработка

В коррелированном подзапросе внутренний подзапрос не может быть реализован немедленно: он ссылается на внешний запрос и выполняется поочередно для каждой строки во внешнем запросе.

Внешний запрос отыскивает первое имя в таблице publishers, затем объединяет publisher pub_id с titles.pub_id, затем запрос возвращается в предложение in и titles.type — сравнивается со строкой «бизнес».

Если удовлетворяет критерию, то происходит выбор.

Составить перечень книг с ценами, равными минимальной цене книги:

```
SELECT MIN(PRICE)
   FROM TITLE      (2.99)
```

Получить названия всех книг, продаваемых по этой цене.

```
SELECT NAME, PRICE
  FROM TITLES
  WHERE PRICE = 2.99.
```

С помощью подзапроса:

```
SELECT NAME, PRICE
  FROM TITLES
  WHERE PRICE =
        (SELECT MIN (PRICE)
         FROM TITLES)
```

Подзапрос, возвращающий единственное значение, начинается с простого оператора сравнения.

5.2 Команды манипулирования данными

После создания БД, необходимо её заполнить. Для заполнения используется оператор INSERT, который позволяет добавлять строки с помощью ключевого слова VALUES или с помощью оператора SELECT.

```
INSERT INTO имя_таблицы [(столбец1 [,столбец2] . . .)]
VALUES (константа1 [, константа2] . . . )
```

Значения нужно вводить в том порядке, в котором определялись столбцы при создании таблицы. Если данные добавляются не во все столбцы, то их нужно доопределить. Если в столбце должно быть значение, либо оно должно быть определено по умолчанию, либо допускать нулевое значение. В этом случае порядок перечисления столбцов может быть любым.

При использовании оператора SELECT в команде INSERT синтаксис несколько изменяется.

```
INSERT INTO имя_таблицы [(вставляемый список столбцов)]
SELECT список столбцов
FROM список таблиц
WHERE условия
```

Оператор SELECT позволяет взять данные из нескольких таблиц и вставить их в другую таблицу. Применяя данный оператор, необходимо помнить, что вставляемые столбцы должны иметь тип соответствующий типу столбца в который значение вставляется или же система должна уметь автоматически производить преобразование.

Последовательности столбцов в таблицах должны быть согласованными. Если они не согласованы, операция INSERT либо не выполниться, либо выполниться не полностью. И при этом данные могут быть размещены в неверных столбцах.

Изменение существующих данных производится с помощью команды UPDATE.

UPDETE имя_таблицы
SET имя_столбца = выражение
[WHERE условие]

В каждом операторе модифицируется только одна таблица. Если добавляемое значение имеет неверный тип или нарушаются другие ограничения на целостность, система обычно запрещает выполнение команды и выдает сообщение об ошибке. В предложении SET определяются столбцы и задаются их новые значения.

Предложение WHERE в операторе UPDATE определяет строки, которые будут изменяться. Предложение WHERE в операторе UPDATE аналогично предложению WHERE в операторе SELECT, и так же может включать в себя подзапросы к одной или нескольким таблицам.

Удаление данных осуществляется с помощью команды DELETE

DELETE FROM имя_таблицы
WHERE условие

В предложении WHERE определяются строки подлежащие удалению. При отсутствии предложения WHERE из таблица удаляться все строки.

6. ФУНКЦИИ СУБД

Функции системы управления базой данных можно разделить на функции пользователя, системные функции и функции управления. Функции пользователя представляют собой функции, непосредственно предлагаемые пользователю. Это поиск данных, удовлетворяющих определенным условиям, обновление данных, расширение данных и т.д. Эти функции реализуются с помощью языка манипулирования данными (или вспомогательного языка данных).

К системным функциям относятся такие, как определение приоритетности, возврат и восстановление данных в связи со сбоем, обеспечение защиты, сжатие данных, интерфейс с операционной системой др.

К функциям управления относятся составление той или иной схемы, задание размещения базы данных на диске, перемещение данных для повышения эффективности использования памяти и т.д.

Архитектура СУБД — совокупность основных ее функциональных компонентов, а также средств обеспечения их взаимодействия (интерфейсов) друг с другом, с пользователями и системным персоналом.

6.1 Функции управления, обеспечение абстракции данных

Одной из наиболее важных функций СУБД, которая оказала решающее влияние на формирование подхода к архитектуре систем, является обеспечение абстракции данных. Механизм абстракции данных, предоставляемых СУБД, служит средством поддержки независимости способов видения БД различными группами пользователей, обеспечивает независимость данных.

СУБД позволяет работать с БД без знания конкретного способа размещения данных в памяти ЭВМ.

Обычно различают три уровня описания данных: концептуальный уровень, логический уровень, физический уровень.

На первом, концептуальном уровне, описываются элементы данных и взаимосвязи между ними, которые являются отражени-

ем объектов и связей реального мира интересующей предметной области без указания способов физического хранения. Здесь при построении концептуальной модели выполняют анализ данных, структуризацию их и связи между ними, а именно имена объектов и элементов данных, описаний, атрибутов, сложность объектов, важность, продолжительность хранения и так далее.

На втором, логическом уровне описывается, как выбранные объекты и связи отражаются в структуре базы данных. То есть это версия концептуальной модели, обеспечиваемая СУБД, которая поддерживает одну из моделей — реляционную, сетевую, иерархическую.

На третьем, физическом уровне описывается, как логическая структура записей будет представлена в памяти посредством методов доступа и методов размещения.

Пользователь через программу или непосредственно с терминала посылает в базу данных запрос на обработку. Этот запрос, записанный на каком-то языке манипулирования данными (ЯМД) и дополненный необходимой информацией из описания этих данных (например, информацией о структуре), приведенного в словаре данных, пересылается в систему управления БД.

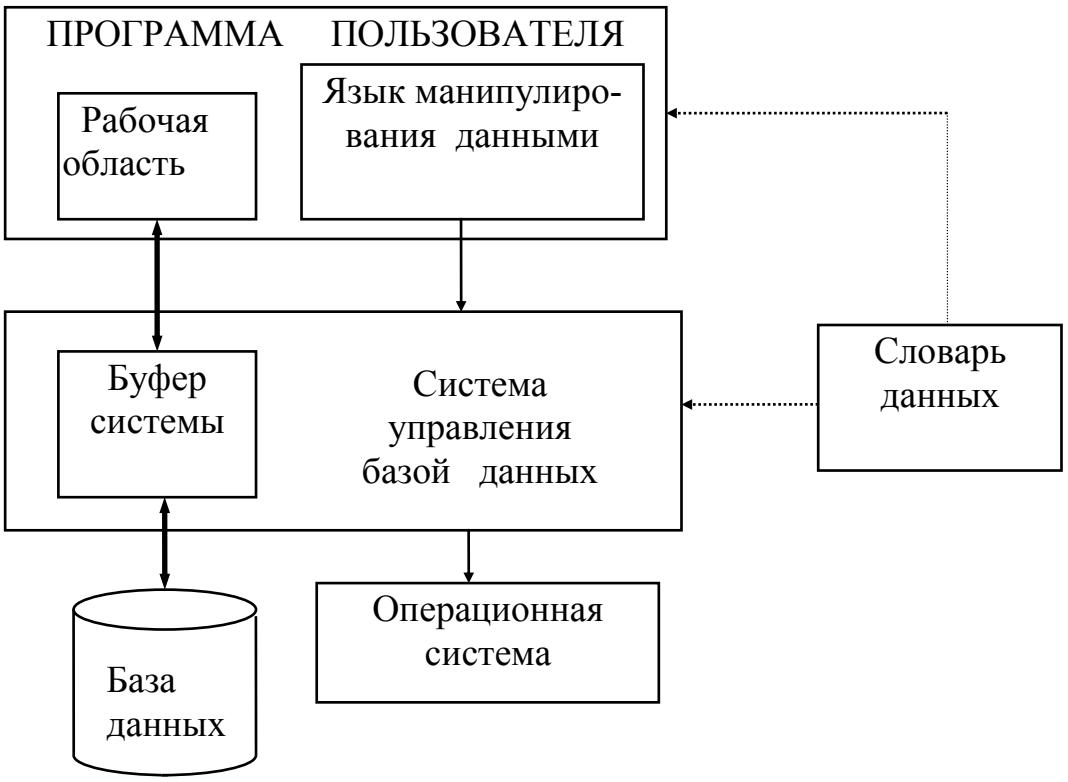


Рис. 21 — Структура системы базы данных

СУБД анализирует это требование, запрашивает из словаря данных информацию о наличии необходимых данных, об их физической структуре и т.д. и передает запрос в форме требования на физический доступ к данным в операционную систему.

Операционная система осуществляет доступ к запрашиваемым данным или другим запоминающим устройствам, передает результат в СУБД. СУБД подвергает результат более детальной обработке и выполняет такие операции над затребованными данными, как пересылка в рабочую область пользователя, переписывание базы данных и т.д.

Из этого явно прослеживается трехуровневая схема представления данных.

6.2 Методы размещения данных

ОЗУ и внешняя память обладают различными возможностями.

Логическая запись состоит из отдельных элементов, связанных отдельными отношениями, которые имеют многоуровневую структуру.

Нижний уровень: числа, символы, логические данные, знаки — эти данные читаются целиком, имеют тип и определенную форму представления в оперативной памяти.

Поле записи описывает атрибут объекта. Имеет наименование и значение.

Группа данных — поименованная совокупность элементов данных рассматриваемая как единое целое. В качестве элемента данных может иметь другую группу данных. Кроме того, может содержать служебную информацию, такую как уникальные ключи, метки, ссылки, указатели.

Логическая запись — поименованная совокупность полей и групп данных. Основная единица обработки информации. Описывает объект. Поля записи могут располагаться друг за другом, а могут структурироваться в сложный массив с нелинейными связями. Структуризация — одна из основных концепций БД.

Отдельные логические записи формируются в информационный массив, хранящейся на ВЗУ, т.е. файл.

Данные в БД хранятся во внешней памяти на магнитных лентах, дисках и т.д. При обработке данные порциями пересылаются из внешней памяти в оперативную. Физическая организация БД влияет на быстродействие системы и в частности на такой параметр, как время отклика.

Время отклика — это промежуток времени между запуском операции БД и получением результата.

В каждой СУБД по-разному организованы файлы и доступ к ним. Как правило, СУБД используют файловые структуры и системы управления файлами, являющиеся частью операционных систем. Поскольку операционные системы предназначены для традиционной обработки файлов, они неудобны и малоэффективны при использовании СУБД. Поэтому в новых системах СУБД сами управляют размещением данных на внешних носителях.

Физическое устройство, записывающее данные на диск, называется ***дисководом***.

Каждый дисковод содержит один пакет дисков.

Пакет дисков состоит из нескольких дисков, нанизанных на вал. Данные записываются на дорожки (концентрические окружности). На одном диске может быть несколько сотен тысяч дорожек. Множество дорожек, расположенных на разных дисках, но имеющих одинаковый размер, называется ***цилиндром***.

Адрес записи на диске обычно состоит из номера цилиндра, номера поверхности и номера блока.

Физическая запись или ***блок*** — это наименьшая единица данных, имеющая физический адрес на диске. Каждая дорожка состоит из множества блоков. Блок может содержать одну или несколько логических записей.

С точки зрения пользователя, ***файлом*** называется поименованная линейная последовательность записей, расположенных на внешних носителях.

Под ***организацией файла*** понимается структура файла определенная в терминах его компонентов и способа их размещения в памяти.

Всякая организация файла обеспечивает один или несколько методов доступа. В настоящее время отсутствует общепринятая классификация организации файла. И обычно она определяется через совокупность обеспечиваемых ею методов доступа.

Доступ — считывание или запись данных с указанием того, меняется ли при этом содержимое файла.

Метод доступа — алгоритм хранения и поиска записей в файле данных или БД.

Для методов доступа известны два критерия эффективности:

1) эффективность доступа — величина обратная среднему арифметическому числу обращений для выборки запроса конкретной записи;

2) эффективность хранения — величина обратная среднему числу байтов поля вторичной памяти, необходимого для хранения одного байта исходных данных.

Существует три основных способа физической организации в запоминающих устройствах: последовательная, прямая, индексная.

6.2.1 Последовательный метод

Хранит физические записи в порядке их поступления. Каждая очередная запись размещается на свободном месте сразу же за последней записью файла. При размещении не существует никакой связи между ключом или идентификатором логической записи и ее местонахождением на носителе. Эффективность доступа очень низкая, так как для нахождения нужной записи необходимо просмотреть все записи БД. Метод используется в основном для восстановления и хранения данных. Эффективность использования памяти стремится к 100%. В процессе последовательной обработки записи обрабатываются в порядке их физического размещения на носителе. Так же как при последовательной обработке, каждая запись обязательно просматривается, но желательно все операции, которые необходимо выполнить над записями, сгруппировать так, чтобы каждая читаемая запись обрабатывалась полностью, и не требовалось ее повторного чтения из файла.

6.2.2 Прямой метод доступа

Самый быстрый метод доступа. Чаще всего в БД необходим поиск по ключу (первичному, возможному, внешнему). Во всех случаях известно значение ключа, но неизвестен номер записи,

который соответствует этому ключу. В этом случае необходимо построить линейную функцию NZ = F(k) которая по значению ключа вычислит номер записи файла. Устанавливает взаимно-однозначное соответствие между ключом записи и ее физическим адресом. Не требует упорядочения значений ключей физических записей. Применяется как для хранения, так и для поиска. Эффективность доступа всегда равна единице, а эффективность хранения зависит от плотности ключей.

Найти взаимно-однозначное соответствие удается не всегда. Часто бывает, что значения ключа разбросаны по нескольким диапазонам и существуют недопустимые значения ключа.

В этих случаях применяют различные методы хеширования (рандомизации). Метод основан на алгоритмическом определении адресов физической записи (используются таблицы) по значениям ключей (или некоторым его характеристикам). Каждый заносимый в хэш-таблицу элемент имеет особый ключ, а само занесение осуществляется с помощью хэш-функции (Fk(k)=k(mod), где k ключ), отображающей ключи на множество целых чисел, которые лежат внутри диапазона адресов таблицы. Функция должна обеспечивать равномерное распределение ключей по адресам таблицы. Но отображение не является взаимно-однозначным. Двум ключам может соответствовать один и тот же адрес. Если ячейка с этим адресом уже занята, то проверка адресов осуществляется до тех пор, пока не найдется свободная (при записи) или искомая (при поиске). Разным ключам, может соответствовать одно значение хэш-функции (один адрес). Подобные ситуации называются коллизиями. Значения ключей, которые имеют одно и тоже значение хэш-функции, называют синонимами. При организации доступа с использованием хэш-функции необходимо:

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

6.2.3 Индексно-последовательный метод

В основе этого метода лежит организация по типу многоуровневого справочника. Каждая логическая запись индексно-

последовательного файла должна содержать атрибут, являющийся ключом. Записи индекса группируются в блоки и их также можно индексировать. Получается иерархическая структура. Если файл очень большой, то применяют несколько ключей для индексации. Эффективность хранения зависит от числа уровней индексации, размещения в памяти, числа записей в БД. Эффективность доступа зависит от величины и частоты обращаемости к БД.

6.2.4 Индексно-произвольный метод доступа

При этом методе доступа записи хранятся в произвольном порядке. Создается отдельный файл (индексный), состоящий из значений ключа и адресов хранимых записей. Индексный файл упорядочен. Между ключом и физической записью установлено взаимно однозначное соответствие. Если ключ состоит из большого количества полей, то объем индексного файла может быть очень большим.

6.3 Системные функции. Обеспечение сохранности и секретности

Помимо задачи обеспечения независимости данных на СУБД возлагаются и другие функции. Это, прежде всего, задачи обеспечения сохранности и секретности данных.

Эти функции тесно связаны между собой, но существует и разница. Под обеспечением сохранности понимается защита данных от:

– непреднамеренного доступа к данным и возможного их искажения со стороны пользователей;

– при сбоях аппаратуры или программных средств.

Это внутренние задачи, т.к. они связаны с обеспечением нормального функционирования БД. Чтобы избежать первой ситуации, вводятся пароли: только для чтения. Во втором случае предполагается специальный режим хранения машинных носителей с данными, экранировку и т.д. Необходимый уровень дублирования данных для последующего восстановления на рабочих носителях в случае сбоя.

Под функцией секретности понимается защита данных от преднамеренного доступа пользователей. В этом случае данные в БД разделяются на общедоступные и секретные. Секретные данные (секретную информацию) можно получить только после специальной алгоритмической обработки. Основной мерой защиты является в этих случаях ввод паролей и идентификация пользователей.

6.4 Обеспечение целостности

Обеспечение целостности базы накладывает ограничения (называемые правилами целостности) на отношения. Функцией системы обеспечения целостности является недопущение противоречий в данных относительно этих условий. Противоречия могут возникнуть в результате ошибок пользователей при вводе данных, путем модификации базы данных, а также при незавершенных операциях модификации по причине сбоев в системе. При этом требуется восстановление данных. При длительной и сложной модификации базы данных накладываемые условия в ходе модификации могут не выполняться.

Проблема целостности заключается в правильности данных в базе данных в любой момент времени и касается защиты данных от непреднамеренных ошибок и их предотвращения.

Поддержка целостности в реляционной модели данных в ее классическом понимании включает в себя три аспекта:

- поддержка структурной целостности;
- поддержка языковой целостности;
- поддержка ссылочной целостности.

Реляционная СУБД работает только со структурой данных типа реляционное отношение. Необходимо поддерживать правила соответствующие реляционной таблице:

1. В таблице нет одинаковых кортежей.
2. Столбцы соответствуют атрибутам отношения.
3. Всегда есть первичный ключ.
4. Каждый атрибут имеет уникальное имя.
5. Порядок строк в таблице произвольный.

6. Два отношения, отличающиеся только порядком следования столбцов считаются одинаковыми.

Проблемы структурной целостности необходимо рассмотреть в случае NULL значений, т.е. таких значений которые на данный момент не определены, но могут быть определены в любое время.

При появлении NULL значений не действуют стандартные правила сравнения: одно неопределенное значение не равно другому неопределенному значению. Для выявления равенства значений в этом случае используют предикаты типа:

<имя атрибута> is NULL

<имя атрибута> is not NULL

Если в данном кортеже указанный атрибут примет неопределенной значение, то предикат is NULL принимает значение true, а предикат is not NULL значение — false. Таблица истинностей логических операций изменилась, поскольку из двузначной превратилась в трехзначную.

Таблица 2

a	b	¬ a	a & b	a | b
1	1	0	1	1
1	0	0	0	1
1	–	0	–	1
0	1	1	0	1
0	0	1	0	0
0	–	1	0	–
–	1	–	–	1
–	0	–	0	–
–	–	–	–	–

В таблице 2 кодирует значение TRUE, 0 — FALSE, – — неопределенное значение.

Поддержка языковой целостности состоит в том, что реляционная СУБД должна обеспечивать языки описания и манипулирования данными не ниже стандарта SQL. То есть не должны быть доступны низкоуровневые средства манипулирования данными, не соответствующие стандарту. Поэтому доступ возможен с использованием SQL.

Ссылочная целостность. Означает обеспечение одного из заданных принципов взаимосвязи между экземплярами кортежей взаимосвязанных отношений.

- Кортежи подчиненного отношения уничтожаются при удалении кортежа основного отношения, связанного с ним.
- Кортежи основного отношения модифицируются при удалении кортежа основного отношения связанного с ним, при этом на месте родительского отношения ставится неопределенное NULL значение.

Структурная, языковая и ссылочная целостность определяет правила работы СУБД с реляционными структурами данных. Требования поддержки трех видов говорят о том, что каждая СУБД должна уметь это делать, а разработчики приложений должны их учитывать.

Но для некоторых ограничений, которые связаны с содержанием базы данных, требуются другие методы. Например, при рассмотрении БД «Отдел кадров» ряд бизнес правил не может быть поддержаны с помощью вышеперечисленных методов.

Например.

На работу преподавателем принимаются граждане, имеющие диплом установленного образца.

Каждый работник должен иметь прописку.

Эти методы сведены в поддержку семантической целостности. Семантическая поддержка обеспечивается двумя путями: декларативным и процедурным.

Декларативный путь связан с наличием в рамках СУБД механизмов обеспечивающих проверку и выполнение ряда декларативно заданных правил-ограничений (бизнес правил).

Существуют следующие виды декларативных ограничений целостности:

- Ограничение целостности атрибута:
  - значение по умолчанию;
  - задание обязательных и необязательных значений;
  - задание условий на значение атрибутов.
- Ограничения целостности, задаваемые на уровне домена; Используются, если в базе присутствуют несколько столбцов разных отношений, которые принимают значения из одного и то-

го же множества допустимых значений. Некоторые СУБД поддерживают доменную структуру и разрешают отдельно определять домены, задавать тип данных для каждого домена и бизнес правила для них. Удобно, если БД большая, содержит сотни отношений. В этом случае поменять условия, задаваемые на домен проще, чем просматривать все атрибуты и менять условия на них.

- Ограничения целостности, задаваемые на уровне отношения.

Например, выразить требования на наличие хотя бы одного телефона — рабочего или домашнего в базе данных «Контакты».

- Ограничения целостности, задаваемые на уровне связи между отношениями: задание обязательности связи, принципов каскадного удаления и каскадного изменения данных. Эти виды ограничений могут быть выражены заданием обязательности или необязательности внешних ключей.

6.5 Функции пользователя. Актуализация данных

Важное место в СУБД занимают процедуры актуализации (обновления) данных. Актуализация, как правило, возлагается на прикладные программы пользователя. Однако при одновременной работе нескольких пользователей с базой данных может возникнуть конкуренция за выполнение операций над одними и теми же отношениями, поскольку каждый пользователь, обрабатывая базу, совершает транзакцию. То есть его действия не должны прерываться. Поэтому при проектировании БД необходимо предусмотреть механизмы захвата или метки времени.

7. ТРАНЗАКЦИИ

Транзакция (transaction) представляет собой логическую единицу работы СУБД. Управление транзакциями обеспечивает интерпретацию некоторого блока операций как неделимого целого. Управление транзакциями гарантирует, что-либо будут выполнены все операции из блока, либо не будет выполнено ни одной. На основании этого тезиса можно выделить следующие свойства транзакций:

Атомарности. Транзакция должна быть выполнена целиком или не выполнена вовсе.

Согласованности. По мере выполнения транзакции данные переходят из одного согласованного состояния в другое, т.е. транзакция не разрушает взаимной согласованности данных.

Изолированности. Конкурирующие за доступ к БД транзакции выполняются изолированно друг от друга, но для пользователя это может выглядеть так, что они выполняются одновременно.

Долговечности. Если транзакция успешно завершена, то изменения данных, которые были произведены, не могут быть потеряны ни при каких обстоятельствах.

Транзакции необходимы для устранения конфликтов пользователей, пытающихся одновременно обращаться к одним и тем же данным, и восстановления данных после сбоя или отказа системы.

Источниками отказов могут быть:

- Системные ошибки. Системные ошибки возникают в случае, если система вошла в нежелательной состояние такое, например, как взаимоблокировка, не позволяющая нормально продолжить выполнение программы. Такой отказ может приводить, а может не приводить к повреждению файлов данных. Принцип блокировки заключается в том, что когда пользователь выбирает данные из БД, СУБД автоматически запрещает к ним доступ, чтобы какой-либо пользователь не мог провести над ними операции обновления. Существует много видов блокировок. В основном операции блокирования и разблокирования производятся автоматически, но в некоторых системах пользователи могут управлять определенными аспектами блокировок, например с

помощью команд SQL они могут выбрать уровень применения блокировки (уровень строки или уровень таблицы).

- Отказы оборудования. Два наиболее часто встречающихся отказа: ошибка диска (авария головки), потеря способности передачи данных через сеть.
- Логические ошибки. Плохие данные или их отсутствие часто является причиной, не позволяющей нормально продолжить выполнение программы.

В стандарте ANSI/ISO SQL включены операторы фиксации транзакции и отката транзакции COMMIT, ROLLBACK.

COMMIT означает успешное завершение транзакции, все изменения принимаются.

ROLLBACK прерывание транзакции, отмена выполненных действий сделанных в БД в рамках этой транзакции.

Новая транзакция может начаться непосредственно после оператора ROLLBACK.

Между командами SQL, отмечающими начало и конец транзакций, может помещаться любое количество операторов SQL. В зависимости от синтаксиса, который поддерживает ваша система, это может выглядеть примерно так:

[оператор начала транзакции]
SQL оператор
SQL оператор
SQL оператор
[оператор конца транзакции]

Если какую-то транзакцию необходимо отменить до того, как она будет выполнена — либо по причине какого-то сбоя, либо просто потому, что пользователю что-то не понравилось, — результаты выполнения всех ее операторов, которые успели завершиться, должны быть отменены. Транзакцию можно отменить (или вернуть в исходное положение) с помощью команды транзакции ROLLBACK в любой момент до выполнения команды транзакции COMMIT.

Для сохранения промежуточных состояний, подтверждения или отката транзакции, используется специальный механизм, называемый ***журналом транзакций***. Журнал транзакций обеспечивает надежное хранение данных в БД, возможность восстанов-

ления согласованного состояния БД после любого рода сбоев. В журнале транзакций фиксируются все изменения БД. После сбоя восстанавливается всегда последнее согласованное состояние базы.

Восстановление базы происходит в следующих случаях:

- Индивидуальный откат транзакций (применение ROLLBACK, аварийное завершение работы программы, взаимоблокировка, при параллельном выполнении транзакций).
- Восстановление после потери содержимого оперативной памяти (мягкий сбой) (аварийное выключение питания, неустранимый сбой процессора).
- Поломка ВЗУ (жесткий сбой).

В журнале транзакций, каждая транзакция имеет свой номер. Все изменения помечаются номером транзакции и значениями изменяемых атрибутов, там же сохраняются команды начала и конца транзакции.

8. ТЕХНОЛОГИЯ КЛИЕНТ-СЕРВЕР

Локальное приложение устанавливается на единичном персональном компьютере, там же располагается и БД, с которой работает данное приложение. Однако необходимость коллективной работы с одной и той же БД повлекла за собой перенос БД на сетевой сервер. Приложение, работающее с БД, располагалось также на сервере. Менее характерным был другой способ, заключавшийся в хранении приложения, обращавшегося к БД, на конкретном компьютере пользователей («клиентов»). Основной проблемой при коллективном доступе к данным, в таких случаях, является проблема обеспечения смысловой и ссылочной целостности данных при одновременном изменении одних и тех же данных.

В ходе эксплуатации таких систем были выявлены общие недостатки файл — серверного подхода. Они состоят в следующем:

– вся тяжесть вычислительной нагрузки при доступе к данным ложится на приложение клиента, что является следствием принципа обработки информации в системах «файл-сервер»: при выдаче запроса на выборку информации из таблицы вся таблица БД копируется на клиентское место, и выборка осуществляется на клиентском месте;

– не оптимально расходуются ресурсы клиентского компьютера и сети; например, если в результате запроса нужно получить 2 записи из таблицы объемом 15 000 записей, то все 15 000 записей будут скопированы с файл-сервера на клиентский компьютер; в результате чего возрастает сетевой трафик, и увеличиваются требования к аппаратным мощностям пользовательского компьютера. Потребности в постоянном увеличении вычислительных мощностей клиентского компьютера обусловливается, в данном варианте, не только развитием программного обеспечения как такового, но и возрастанием обрабатываемых объемов информации;

– в БД на файл-сервере гораздо проще вносить изменения в отдельные таблицы, минуя приложения, непосредственно из инструментальных средств; подобная возможность облегчается тем обстоятельством, что, фактически, у локальных СУБД база дан-

ных понятие более логическое, чем физическое, поскольку под БД понимается набор отдельных таблиц, сосуществующих в едином каталоге на диске. Все это позволяет говорить о низком уровне безопасности — как с точки зрения хищения и нанесения вреда, так и с точки зрения внесения ошибочных изменений;

– бизнес — правила (определяют реакцию системы на добавление, изменение, удаление данных и реализуют блокировку действий, которые могут разрушить ссылочную или смысловую целостность БД) в системах файл-сервер реализуются в приложении, что позволяет в разных приложениях, работающих с одной БД, проектировать взаимоисключающие бизнес-правила; смысловая целостность информации при этом может нарушаться.

Приведенные недостатки решаются при переводе приложений из архитектуры файл-сервер в архитектуру клиент-сервер, которая знаменует собой следующий этап в развитии СУБД. Характерной особенностью архитектуры клиент-сервер является перенос вычислительной нагрузки на сервер БД (SQL-сервер) и максимальная разгрузка приложения клиента от вычислительной работы, а также существенное укрепление безопасности данных — как от злонамеренных, так и просто ошибочных изменений.

БД в этом случае помещается на сетевом сервере, как и в архитектуре «файл-сервер», однако прямого доступа к БД из приложений не происходит, функции прямого обращения к БД осуществляет специальная управляющая программа — сервер БД (SQL-сервер), поставляемая разработчиком СУБД.

Взаимодействие сервера БД и приложения-клиента происходит следующим образом: клиент формирует SQL-запрос и отсылает его серверу. Сервер, приняв запрос, выполняет его, и результат возвращает клиенту. В клиентском приложении в основном осуществляется интерпретация полученных от сервера данных, реализация интерфейса с пользователем и ввод данных, а также реализация части бизнес-правил.

Преимущества архитектуры клиент-сервер:

1) большинство вычислительных процессов происходит на сервере; таким образом, снижаются требования к вычислительным мощностям компьютера клиента;

2) снижается сетевой трафик за счет посылки сервером клиенту только тех данных, которые он запрашивал; например, если необходимо сделать из таблицы объемом 15 000 записей выборку, результатом которой будут всего 2 записи, сервер выполнит запрос и перешлет клиенту набор данных из 2 записей;

3) упрощается наращивание вычислительных мощностей в условиях развития программного обеспечения и возрастания объемов обрабатываемых данных: проще и чаще дешевле усилить мощности на сетевом сервере или полностью заменить сервер на более мощный, нежели наращивать мощности или полностью заменять 100–500 клиентских компьютеров;

4) БД на сервере представляет собой, как правило, единый файл, в котором содержатся таблицы БД, ограничения целостности и другие компоненты БД. Взломать такую БД, даже при наличии умысла, тяжело. Значительно увеличивается защищенность БД от ввода неправильных значений, поскольку сервер БД проводит автоматическую проверку соответствия вводимых значений наложенным ограничениям и автоматически выполняет необходимые бизнес-правила: кроме того, сервер отслеживает, уровни доступа для каждого пользователя и блокирует осуществление попыток выполнения неразрешенных для пользователя действий, например, изменения или просмотр таблиц; все это позволяет говорить о значительно более высоком уровне обеспечения безопасности БД и ссылочной и смысловой целостности информации;

5) сервер реализует управление транзакциями и предотвращает попытки одновременного изменения одних и тех же данных;

6) безопасность системы возрастает за счет переноса большей части бизнес-правил на сервер; падает удельный вес противоречащих друг другу бизнес-правил в клиентском приложении, выполняющих разные действия над БД; определить такие противоречивые бизнес-правила в приложениях клиента все еще можно, однако намного труднее их выполнить ввиду автоматического отслеживания сервером БД правильности данных.

При этом схемы использования клиент-серверной технологии приведены на рисунках 22–24. Положительными моментами использования является обилие инструментальных средств. Не-

достатки: высокая загрузка сети, невозможность удовлетворительного администрирования. Различные по природе функции смешиваются в одной программе.

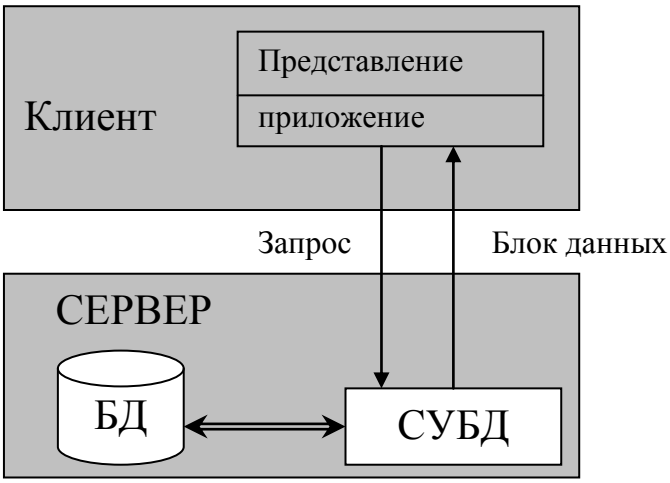


Рис. 22 — Схема непосредственного
клиент-серверного доступа

При взаимной работе коллективом разработчиков с одной базой сложно контролировать взаимную непротиворечивость алгоритмов обработки.

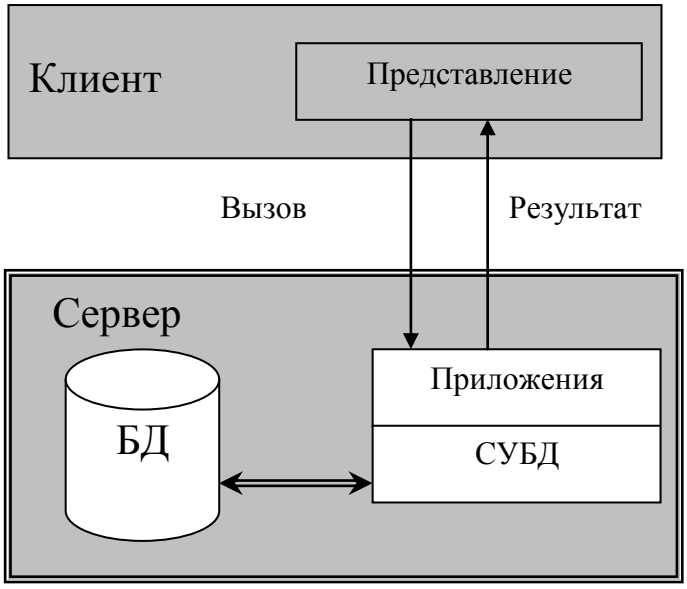


Рис. 23 — Схема клиент-серверного доступа
через приложение на стороне СУБД

Приложение оформляется как набор хранимых процедур SQL и переносится на компьютер — сервер БД. При этом снижается трафик сети, возникает возможность централизованной обработки данных. Когда БД велики, за счет сложного администрирования приводит к потере качества. Процедурное расширение SQL не является полноценным языком программирования. Отсутствуют средства отладки и тестирования хранимых процедур. На практике, на сервере хранятся простейшие процедуры, вся работа по прежнему проводится на компьютере-клиенте. Требуется дополнительная аппаратура и ПО.

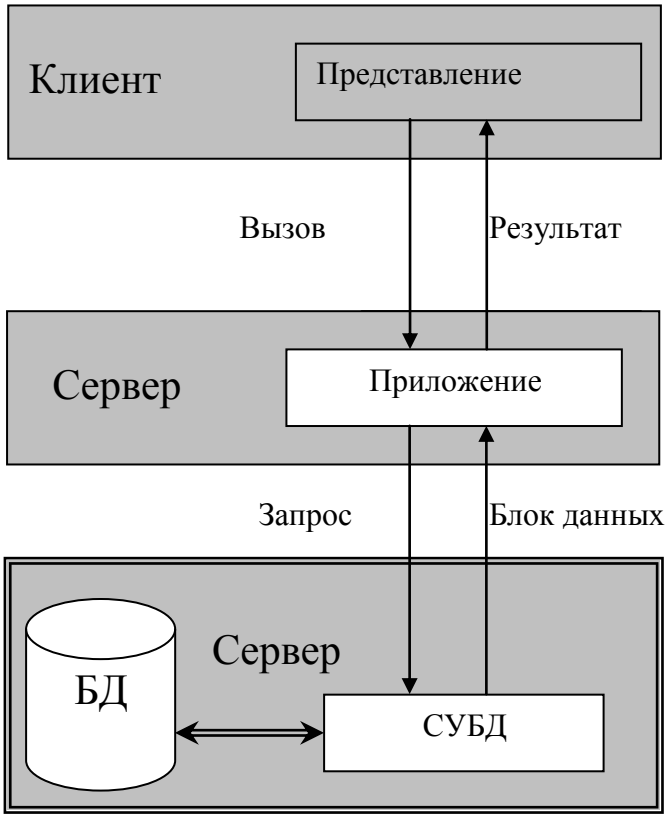


Рис. 24 — Схема трехзвенного
клиент-серверного доступа

В последнем случае: выделение третьего звена — сервера приложений. В этом случае прикладная система независима от СУБД и от клиента. Данная архитектура обеспечивает баланс загрузки и возможность динамической реконфигурации системы. Но при этом дополнительные средства уходят на аппаратуру. Плюс инструментальные средства — Visual Basic, C++, расши-

ренный SQL. Вместе с тем необходимо при разработке требований к библиотеке свободно распространяемых программных средств для организации и проведения автоматизированных лабораторных практикумов в режиме многопользовательского удаленного доступа к сети Интеренет учитывать вопросы безопасности, что очень актуально при проведении лабораторных практикумов с использованием лабораторных стендов.

9. БЕЗОПАСНОСТЬ БАЗ ДАННЫХ

Защита информации в процессе ее сбора, хранения и обработки приобретает исключительно важное значение, поскольку информация представляет собой, чрезвычайно важный ресурс.

Под защитой информации понимается совокупность мероприятий, методов и средств, обеспечивающих решение следующих основных задач:

- Обеспечение целостности информации.
- Исключение несанкционированного доступа к ресурсам ЭВМ и хранящимся в ней программам.
- Исключение несанкционированного использования хранящихся в ЭВМ программ и данных.

В соответствии с первым пунктом необходимо предупредить изменение или разрушение данных при сбоях аппаратуры, программных средств, ошибках в работе сотрудников групп эксплуатации. Методы, обеспечивающие целостность информации, были рассмотрены выше.

На самом элементарном уровне концепции обеспечения безопасности баз данных и управления информацией исключительно просты. Необходимо поддерживать два фундаментальных принципа: проверку полномочий (санкционирование) и проверку подлинности (аутентификация).

Проверка полномочий основана на том, что для каждого пользователя или процесса информационной системы устанавливается набор санкционированных действий, которые он может выполнять по отношению к определенным объектам.

Проверка подлинности означает достоверное подтверждение того, что пользователь или процесс, пытающийся выполнить санкционированные действия, действительно является тем, за кого он себя выдает. Рассмотрим каждый из этих принципов подробнее.

Для проверки полномочий можно построить матрицу безопасности, устанавливающую отношения между всеми пользователями и процессами системы, с одной стороны, и всеми объектами предметной области — с другой. В каждой клетке такой матрицы можно было бы хранить список, содержащий от 0 до n

операций, которые рассматриваемый пользователь или процесс имеет право выполнять по отношению к данному объекту. Всемогущая и неуязвимая система управления безопасностью, которая могла бы поддерживать такую матрицу и принуждать к ее использованию при выполнении любой операции в данной среде, вероятно, гарантировала бы высокий уровень безопасности информации, но дело обстоит значительно сложнее.

Такая базовая модель сама по себе недостаточна для обеспечения хотя бы минимального уровня безопасности.

Пусть существует в системе два процесса. Первый обладает полномочиями на выполнение операций, а второй нет. Если, например, процессу 2 удастся успешно выдать себя за процесс 1, то он сможет выполнять действия и операции, допустимые только для процесса 1 (но не для процесса 2).

В безопасной среде должна поддерживаться какая-либо модель проверки подлинности, которая может обеспечивать надежную верификацию идентификаторов, предъявляемых пользователями или процессами. Проверка подлинности приобрела еще большее значение в условиях массового распространения распределенных вычислений. При существующем высоком уровне использования соединений вычислительных систем необходимо контролировать все обращения к системе.

Проблемы проверки подлинности обычно относят к сфере безопасности телекоммуникаций и сетей, поэтому мы не будем их здесь далее обсуждать и ограничимся следующим замечанием. В целостной системе безопасности компьютерной среды, где исчерпывающая программа защиты информации обеспечивается за счет взаимодействия операционных систем, сетей, систем баз данных и других сервисов, проверка подлинности имеет прямое отношение к безопасности баз данных.

Модель безопасности, основанная на базовых механизмах проверки полномочий и подлинности, не решает таких проблем, как хищения пользовательских идентификаторов и паролей или злонамеренные действия некоторых пользователей, обладающих полномочиями. Например, программист, работающий над учетной системой, имеющей полный доступ к учетной базе данных, может встроить троянского коня в код программы с целью хищения средств или с другими подобными умыслами.

10. ПРОБЛЕМЫ ПРОЕКТИРОВАНИЯ ИНТЕРФЕЙСА

Интерфейс человек — компьютер включает все те аспекты автоматизированной вычислительной системы, с которыми непосредственно соприкасается пользователь.

Общий психологический климат в организации, а также преподнесение сведений о вычислительной системе могут вызвать предубеждение против этой системы задолго до того, как пользователь познакомится с ней практически.

Тело человека — это «механизм», который должен работать в рамках определенных ограничений и допусков. Глазам нужно, чтобы образы имели определенный размер, уровень яркости, контрастности, располагались на удобном расстоянии. Некоторые цвета воспринимаются легче других. Суставы ограничены в движениях, кроме того, нам нужна «подпорка». У нас очень большая долговременная память и очень ограниченная кратковременная (оперативная). Если мы регулярно выполняем какую-то работу, то со временем мы выполняем ее на подсознательном уровне, не перегружая в оперативную память. Быстрый доступ к долговременной памяти. Например, запомнить содержимое полей на разных экранах гораздо труднее, чем на одном.

Люди для всяких действий строили модель деятельности, базирующейся на прошлом опыте человека. Под долговременной памятью часто понимают хранилище различных моделей, на базе которых человек строит свою умственную деятельность. Когда модель не совпадает с реальностью, возникает стресс и как следствие — заболевание и большое количество ошибок при работе. Средства адаптации человека находятся в рамках интерфейса.

Что входит в состав интерфейса?

С точки зрения программного обеспечения в состав интерфейса входят два компонента: набор процессов ввода/вывода; процесс диалога.

Процессы ввода/вывода служат для того, чтобы принять от пользователя и передать ему данные через различные физические устройства. Устройства вывода: дисплеи (текстовая и графическая информация); твердые копии — АЦПУ, лазерные принтеры,

графопостроители; звуковой вывод — синтезаторы (речь), звукогенераторы (музыка); фотографический — видеоинформация.

Устройства ввода: ввод данных людьми — клавиатура (текст), планшеты (графический ввод); автоматический сбор информации — сканеры; позиционирование и выбор — мышь; речевой ввод и машинное зрение. При выборе устройств учитываются следующие факторы:

1) содержание и формат обрабатываемых данных;

2) объем ввода/вывода (влияет на автоматизацию ввода);

3) ограничения, накладываемые пользователем и рабочей средой (например, цех);

4) ограничения, связанные с другими аппаратными и программными средствами.

Разговор — устный обмен информацией между двумя или большим числом людей, осуществляемый посредством слов в соответствии с определенными правилами.

Правила разговора:

1) участники должны понимать язык друг друга;

2) нельзя говорить одновременно: один говорит — другой слушает;

3) информация, которую сообщает оратор, связана, как правило, с предысторией;

4) информация, которой обмениваются говорящие, является последовательностью связных предложений.

Диалог и разговор синонимы, но термин «диалог» используется чаще в применении к компьютерам.

Диалог между человеком и компьютером можно определить как обмен информацией между вычислительной системой и пользователем, проводимый с помощью интерактивного терминала и по определенным требованиям.

Процесс диалога — это механизм обмена информацией.

Задачи диалогового процесса:

1) определение задания, которое пользователь возлагает на систему;

2) прием логически связанных входных данных от пользователя и размещение их в переменных соответствующего процесса в нужном формате;

3) вызов процесса выполнения требуемого задания;

4) вывод результатов обработка по окончании процесса в подходящем для пользователя формате.

Во время диалога обмен информацией между его участниками осуществляется посредством сообщений.

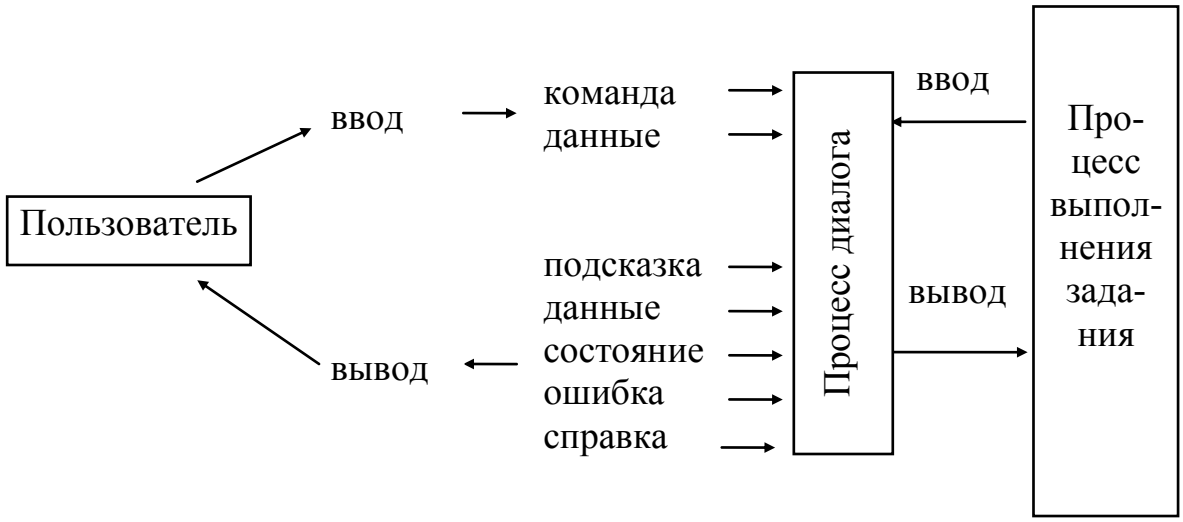


Рис. 25

Подсказка — это выходное сообщение системы, побуждающее пользователя вводить данные.

Сообщение об ошибке — сигнал диалогового процесса о том, что невозможно дальнейшее выполнение работы, так как он или процесс, вызванный для выполнения задания, не может обработать сообщение, введенное пользователем.

Выходные данные — это данные, которые возвращает процесс по окончанию обработки, выданные в форме, удовлетворяющей пользователя.

Сообщение о состоянии системы — это сообщение о том, что произошло или происходит в системе, например, расчет окончен или процент копирования.

Справочная информация — при затруднении пользователя диалоговый процесс может вывести справочную информацию, поясняющую, что делать и почему.

Выделяют четыре структуры диалога: вопрос и ответ; меню; диалог на основе экранных форм; диалог на базе команд.

В первых трех формах диалога ведущим является компьютер, в последней форме — человек. Какая структура лучше, зависит от назначения системы и условий ее применения.

Структура диалога вопрос-ответ построена по типу интервью. Является несколько устаревшей, поскольку появились мощные средства ввода/вывода. Но она достаточно гибка и удобна для пользователя, и от нее не нужно отказываться. В частности эта структура диалога используется в экспертных системах, так как неизвестно какой вопрос необходимо задать следующим.

Меню — одна из популярнейших структур диалога. Такой тип интерфейса стал особенно популярным с появлением манипулятора мышь. Меню в виде блока информации, в виде строки и использование «оконной» техники — традиционный подход. Как правило, меню строятся по иерархическому принципу. Не самый быстрый диалог. Для подготовленного пользователя бывает скучен, так как на одни и те же пункты меню требуют одни и те же данные. Для динамики вводятся «горячие» клавиши, позволяющие выйти в нужный пункт подменю, минуя длинную и нудную последовательность меню. Эта структура хороша для неподготовленных пользователей и при использовании манипулятора типа мышь.

Структура диалога на основе экранных форм — позволяет получить от пользователя сразу всю необходимую информацию. Служащие, как правило, работают со стандартными формами, например счетами, различного рода бланками. При этом учет деятельности требует достаточно стандартного набора данных. Такого типа интерфейс уместно применять там, где есть соответствующая канцелярская форма. Форма, отображаемая на экране должна быть похожа на соответствующую канцелярскую форму. Поля ответов должны быть выделены либо цветом, либо соответствующими символами. При этом необходимо, чтобы последовательность заполнения экранной формы не играла роли. И пользователь мог возвращаться к ответу несколько раз. Конец ввода должен фиксироваться какой-то конкретной клавишей, и обработка формы должна проводиться вся целиком. Эта форма меню обеспечивает высокий уровень поддержки пользователя, работает быстрее, чем предыдущие, используется для заполнения различных бухгалтерских форм, а также для задания параметров запросов в различных базах данных.

Структура диалога на базе командного языка — удобна для ввода управляющих сообщений, обеспечивает широкие воз-

можности выбора в любой точке диалога и не требует иерархических построений как в диалоге типа меню. Диалог в режиме команд работает как телетайп. В обычное время система выдает только подсказку. Чаще всего этот режим используется для работы с операционными системами. Диалог быстрый. При использовании его пользователь должен знать не только команды, но и позиционные параметры, а также основные ключи. Кроме того, многие командные языки поддерживают макросы, которые позволяют увеличивать число команд.

Все структуры диалога при детальном рассмотрении являются теми или иными разновидностями структуры Q&A. Выбор соответствующей структуры зависит от того, насколько квалифицированным будет пользователь, работающий с приложением, и от сферы применения приложения. При проектировании сначала формулируют требования к диалогу, исходя из особенностей приложения, затем выбирают соответствующий диалог. На практике чаще всего используются смешанные формы диалога, с адаптацией к уровню подготовки пользователя.

При проектировании интерфейса для приложения необходимо учитывать следующие факторы:

1) диалог должен быть естественным, то есть пользователь при работе с приложением не должен существенно менять свои привычки;

2) стиль диалога должен быть разговорным;

3) последовательность запрашиваемой информации постоянна и в том порядке, какой соблюдается традиционно. Последовательность — это фактор, на который должен обратить особое внимание разработчик. Последовательность гарантирует, что вводимые коды, например ключевые слова, во всех частях системы будут трактоваться одинаково. Последовательность в размещении данных гарантирует, что пользователь нужную информацию, например сообщение об ошибке, всегда найдет на одном и том же месте экрана;

4) краткость. Диалог должен быть кратким. Система должна требовать от пользователя ввода минимума информации (иметь какие то начальные данные) иначе можно легко ошибиться.

5) поддержка пользователя во время работы с системой — различного рода инструкции для каждого этапа работы с

системой. Характер выдаваемых сообщений об ошибках, подтверждение действий системы, если это необходимо;

6) гибкость диалога — адаптация системы под уровень подготовки пользователя.

WIMP — интерфейс.

Расшифровывается следующим образом:

W[indows] — информация представляется пользователю на экране дисплея в виде нескольких окон;

I[con] — объекты, с которыми система имеет дело, представляется пиктографически в виде икон;

M[ouse] — выборка производится с помощью манипулятора типа «мышь»;

P[op-up] — означает меню, которые автоматически всплывают на экране, или которые пользователь может «вытянуть» (pull down) из строки меню, расположенной в верхней части экрана.

Популярность WIMP-интерфейса в настоящее время объясняется ростом числа персональных компьютеров обладающих:

– достаточной памятью прямого доступа для обеспечения изображения, в несколько раз превышающего размеры физического экрана, и допускающей перекрытие частей экрана;

– экраном с поточечной адресацией средней или высокой разрешающей способности для обеспечения графических изображений;

– адекватной вычислительной мощностью для быстрой «перерисовки» экрана и для обеспечения мультипрограммного режима.

В основу WIMP-интерфейса легли три принципа:

1) все элементы представляются объектами, с которыми возможно «прямое манипулирование», например буксировка манипулятором типа мышь;

2) все действия, выполняемые ЭВМ, немедленно отображаются на экране, т.е. интерфейс эффективно обеспечивает информацию о статусе объекта, подтверждая, что действие выполнено;

3) принцип «рабочий стол» предполагает, что пользователь имеет все необходимые ему источники информации. А именно, различного типа документы: текст, таблицы цифр, графики, ри-

сунки и др. Кроме того, при работе можно менять тип задания (переключаться с электронных таблиц в систему подготовки текстов), обмениваться заданиями между приложениями. При этом доступны такие вспомогательные средства, как электронная почта, калькулятор, часы, записная книжка и др.

Все действия выполняются посредством окон. ***Окно*** — специальная (обычно прямоугольная) область физического экрана, с помощью которой пользователь рассматривает различные аспекты своего взаимодействия с задачей. Идея окна не нова и появилась с того времени, когда появились дисплеи, работающие в постраничном режиме.

В системе управления окнами входные и выходные процессы записывают в виртуальные буферы экрана, а не на физический экран. Это снимает ограничение размеров физического экрана при выводе и допускает многократно перекрывающиеся виртуальные экраны. Отображение производится процессом управления дисплеем, который обеспечивает функции:

- открытия и закрытие окна;
- выборки активного окна;
- перемещения окна относительно его буфера;
- перемещения окна относительно экрана;
- изменения атрибутов окна.

Процесс, управляющий вспомогательными буферами, требуется для управления буферами виртуального экрана, которые могут быть в памяти, на диске или распределяться между ними.

Процесс диалога активизирует систему управления отображением и функции управления буферами через внешние системные вызовы. Пользователь может активизировать их либо через структуры командного языка, либо путем прямого манипулирования. Система прямого манипулирования обращается с элементами данных как с конкретными объектами, которые могут быть физически обработаны такими средствами, как буксировка, указание и выборка, экранные кнопки и затирание.

Прямое манипулирование представляет интерфейс, который естественен и обеспечивает высокий уровень поддержки начинающих пользователей. Однако он может быть избыточным, не-

гибким и чрезмерно регламентированным для опытного пользователя.

Все рассмотренные выше интерфейсы относятся к неинтеллектуальным, так как интерфейс не «понимает» содержания информации, которой он манипулирует. Ему достаточно знать только правила преобразования одного типа информации в другой тип.

Контрольные вопросы

1. Что такое информация, данные, инфологический и датологический аспекты проектирования ПО?
2. Какие разновидности АИС Вы знаете?
3. Дайте определения банка данных, базы данных, системы управления БД.
4. Из каких подсистем состоит АИС?
5. Каким образом можно описать ПО? Дайте определение понятиям: объект, атрибут, связь.
6. Какого типа связи вы знаете, примеры?
7. Дайте понятие модели данных, модели знаний, каково их различие.
8. Иерархические модели данных. Приведите пример.
9. Сетевые модели данных. Приведите пример.
10. Реляционные модели данных. Дайте определение отношения, кортежа, атрибута, ключа.
11. Реляционное исчисление. Дайте определение предиката, предиката и отношения.
12. Какие операции над предикатами Вы знаете?
13. Реляционная алгебра. Определите правила реляционной таблицы.
14. Концептуальная модель.
15. Языковая модель (ЯОД, ЯМД, базовые языки, включающие языки).
16. Проблемы разработки интерфейса (состав, факторы влияющие на разрабатываемый интерфейс и т.д.) Задачи диалогового процесса (типы сообщений, структуры диалога, характеристики диалога).

17. Типы интерфейса (A&Q, экранные формы, меню, командный, WIMP).

18. Методы доступа к внутренней модели (последовательный, индексно последовательный, индексно произвольный прямой, хеширования).

19. Обеспечение независимости данных(физическая, логическая модели).

20. Обеспечение целостности данных, сохранности, секретности, актуализации.

КУРСОВОЕ ПРОЕКТИРОВАНИЕ

Выполнение курсовой работы позволяет закрепить полученные навыки.

При проектировании необходимо ответить на ряд вопросов: Откуда поступает информация и в каком виде? Как она вводится в систему и кто этим занимается? Как часто она изменяется? Какие параметры системы наиболее критичны с точки зрения реакции системы и ее надежности. Необходимо также изучить все бумажные информационные формы, а также файлы.

Процесс создания структуры базы данных состоит из следующих этапов:

1. Исследование предметной области и выделение основных задач, для решения которых предназначена база данных.

2. Создание списка объектов вместе с их свойствами и атрибутами.

3. Определение для каждого объекта атрибута или группу атрибутов по которому однозначно можно идентифицировать экземпляр объекта (строку в таблице).

4. Определение идентифицирующих и не идентифицирующих связей между объектами.

5. Создав структуру БД, проанализируйте ее с точки зрения правил нормализации.

Создайте структуру и напишите несколько SQL запросов к спроектированной базе.

После проектирования структуры, продумайте функциональные возможности приложения. Создайте функциональную модель приложения и приступайте к кодированию.

Задания на курсовую работу

При выполнении курсовой работы необходимо:

- Исследовать заданную предметную область, определить объекты, выбрать существенные атрибуты. Задать бизнес правила.
- Разработать логическую модель. Задать первичные и внешние ключи.
- Провести нормализацию полученной базы до уровня Бойса–Кодда.

- Построить модель FA-уровня.
- Объяснить выполненные преобразования.
- Провести исследование полученной модели, задав несколько сложных SQL запросов к полученной модели.
- Написать приложение, отрабатывающее заданные бизнес правила.
- Предметную область для проектирования выбрать по стандартным правилам.

Номера вариантов

1. Деятельность ресторана/кафе/кулинарные рецепты (меню, блюда и рецепты их приготовления с произвольным количеством возможных продуктов, возможность расчета стоимости блюда по ценам составляющих продуктов, верификация наличия продуктов, замена аналогами).

2. 1С Торговля (поступление товара на склад по накладной, продажа товара со склада выпиской счет фактур, учет товаров на складах).

3. Телефонный справочник S09 (поиск по телефону, адресу и владельцу, возможность двух режимов работы: пользовательском — когда возможна справочная работа с базой данных и административном — когда возможно дополнение/изменение информации).

4. Расчет заработной платы работников организации (начисления, удержания, ставки, стаж, тарифная сетка по стажу, табель, районный коэффициент, детские, отчисления и др.).

5. Железнодорожные кассы / авиакассы (продажа билетов, поиск/составление маршрутов, выдача информации о наличие билетов, предварительный заказ билетов/бронирование мест).

6. Работа поликлиники (расписание работы врачей, запись на прием, выбор из возможных вариантов времени и даты клиенту посещения врача, фиксация приема у врачей, оплата услуг, назначенные процедуры и анализы и др.).

7. Деятельность любого магазина/розничной/оптовой точки (сеть магазинов/торговых точек, прайс-лист по текущему состоянию на складе, проверка наличия того или иного товара как внутри

одного магазина/торговой точки так и в нескольких, продажа товаров, учет товаров на складах, заказ товаров, история цен и др.).

8. Аптека (аналогично с магазином/торговой точкой, но, например, предусмотреть замену лекарств аналогами).

9. Туристическое агентство (оформление и продажа путевок в различные страны с различным сервисом и длительностью).

10. Библиотека (картотека, выдача книг читателям и пр.).

11. Работа с электронными картами/со счетом в банке (срок действия карты/счета, номер и ПИН код, валидность карты/счета, контроль остатка, осуществление операций по картам/счетам и др.).

12. Видеотека/CDтека (можно предусмотреть прокат, продажу и др.).

13. Автомагазин (аналогично с магазином/торговой точкой, но, например, предусмотреть контроль пробега, год выпуска, марки, БУ и др.).

14. Страховая компания (выдача полисов, страхование имущества и др.).

15. Электронный магазин (имитация электронного магазина: авторизация входа, просмотр каталогов товаров, отображение товаров на складе, помещение в корзину/удаление из корзины, проверка валидности карт и пр.).

16. Электронный деканат (часть его деятельности).

17. Отдел кадров (ведение приказов о принятии на работу, увольнении, взысканиях, поощрения, перевод на другую должность, стаж).

18. Агентство недвижимости (аренда, продажа квартир, поиск подходящих вариантов и пр.).

19. ГИБДД (учет зарегистрированных и угнанных автомобилей, выдача прав, техосмотр и пр.).

20. 1С производство/производственный процесс (проекты/разработки, детали, материалы, полуфабрикаты, поставщики, выпуск продукции и пр.).

21. Реализация электронных тестов (список тестов, авторизация тестируемого, оценка, возможность повторного тестирования с отображением новой оценки и прогресса в сравнении с предыдущим тестом и пр.).

СПИСОК ЛИТЕРАТУРЫ

1. Кагаловский М.Р. Технология баз данных на персональной ЭВМ. — М.: Финансы и статистика, 1992. — 224 с.

2. Ревунков Г.И., Самохвалов Э.М., Чистов В.В. Базы и банки данных и знаний. — М.: Высшая школа, 1992 — 368 с.

3. Дейт К. Введение в системы баз данных. — М.: Наука, 1980. — 463 с.

4. Гэри Хансен, Джеймс Хансен Базы данных: разработка и управление: Пер. с англ. — М.: ЗАО «Издательство БИНОМ», 1999. — 704 с.

5. Боуман Джудит, Эмерсон Сандра, Дарновски Марси Практическое руководство по SQL / 3-е издание.: Пер. с англ. — К.: Диалектика, 1997. — 320 с.

6. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год: Пер. с англ. / Под ред. и с предисл. М.Р. Кагаловского. — М.: Финансы и статистика, 1999. — 479 с.

7. Базы даны: модели, разработка реализация / Т.С. Карпова. — СПб.: Питер, 2001. — 304 с.

ПРИЛОЖЕНИЕ А

ПРИМЕР ОТЧЕТА ПО КУРСОВОЙ РАБОТЕ

Пример титульного листа

Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

Проектирование учебно-исследовательской базы данных
«название проекта»

Пояснительная записка к курсовой работе по дисциплине
«Базы данных»

ФВС КП. 00007-01 81 01

Студент гр. №
_____ФИО

«____» _____200_

Руководитель
_____ ФИО
«____» _____200_

Город – 200_

Пример

СОДЕРЖАНИЕ

Техническое задание

Прилагается
Дискета 3,5″ В конверте
на обороте
обложки

Пример реферата

РЕФЕРАТ

Курсовая работа 31 с., 5 источников, 4 приложения.

ПРОКАТ ДИСКОВ, СУЩНОСТЬ, СВЯЗЬ, АТРИБУТ, КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ, IDEF1X, ФИЗИЧЕСКАЯ РЕАЛИЗАЦИЯ.

Объектом исследования являются бизнес — процессы в фирме по прокату компьютерных дисков.

Цель работы — описание структуры реляционной базы данных «Прокат дисков», и разработка прикладной программы, предназначенной для ее информационной поддержки.Программа тестировалась на компьютере Intel Celeron 850MHz, ОЗУ 384Mb, ОС Windows XP.

Пояснительная записка выполнена в текстовом редакторе Microsoft Word 2000 и представлена на дискете 3,5″.

Пример основной части

1. *ВВЕДЕНИЕ*

Во введении пишется общая часть по проекту.

2. *ПРОЕКТИРОВАНИЕ ИНФОЛОГИЧЕСКОЙ МОДЕЛИ ДАННЫХ*

Структуризация предметной области

Пункт проката дисков имеет фонд лазерных дисков и получает прибыль от проката и продажи отдельных экземпляров

Клиентами фирмы могут быть как физические, так и юридические лица. Клиент может приобрести диск, но при том его стоимость превышает рыночную цену. Прокат диска осуществляется следующим образом: клиент оставляет залог, размер которого равен целому числу m, умноженному на стоимость суточного проката. Все клиенты, желающие взять диск напрокат, проходят процедуру регистрации, указывая в качестве обязательных данных фамилию и имя; остальные поля служат для анализа интересов клиентуры фирмы.

Необходима также регистрация факта выдачи каждого диска, т.е. создание учетной записи с указанием фамилии клиента, названия диска и даты выдачи. По истечении m дней диск считается проданным и учетная запись удаляется.

При возврате диск проверяется на дефектность, соответствующий факт фиксируется.

По желанию оператора составляется отчет с указанием размера выручки, количества списанных дисков, названий дисков, пользующихся наибольшим спросом, а также дисков, отсутствующих в каталоге, но заказанных клиентами, и другой информацией.

В случае, если клиент заказывает диск, количество экземпляров которого указано равным нулю, клиент может оформить требование и тогда по возвращении взятого на руки экземпляра он должен быть выдан клиенту, оформившему требование в порядке очереди.

В настоящей работе реальная предметная область ограничена автоматизацией обслуживания клиентов и не рассматривает вопросы обновления фонда.

Основные компоненты бизнеса:

- клиенты;
- каталог дисков;
- оператор;
- учетные записи;
- требования.

Основные бизнес-процессы:

- выдача экземпляров дисков;
- возвращение дисков в каталог;
- проверка диска на дефектность;
- оформление требований;
- уведомление о необходимости выдачи возвращенного диска оформителю требования;
- преобразование требование → учетная запись.

База данных должна поддерживать накопление и хранение информации об основных компонентах бизнеса и автоматизированное выполнение бизнес-процессов.

Формализованное описание задачи

Бизнес-правила:

- устаревшие учетные записи удаляются
- максимальный срок проката — размер залога, деленный на стоимость суточного проката;
- оформить заказ на прокат может только зарегистрированный клиент;
- заказать диск, отсутствующий в каталоге, может любой клиент;
- при удалении учетной записи происходит проверка на наличие требований на этот диск;
- при обновлении каталога данные о списанных дисках уничтожаются.

Перечень вводимой информации:

- фамилия, имя и отчество клиента;

- электронный адрес клиента;
- род занятий;
- возраст;
- категория диска;
- жанр;
- название диска;
- количество экземпляров диска;
- дата выдачи копии в прокат;
- обусловленная дата возврата копии;
- признак дефектности диска;
- названия новых дисков;
- данные для вывода в отчет;
- цена дня проката копии;
- размер залога.

Сведения об имеющихся в фонде дисках могут быть доступны клиентам фирмы.

3. *ПРОЕКТИРОВАНИЕ ЛОГИЧЕСКОЙ МОДЕЛИ ДАННЫХ*

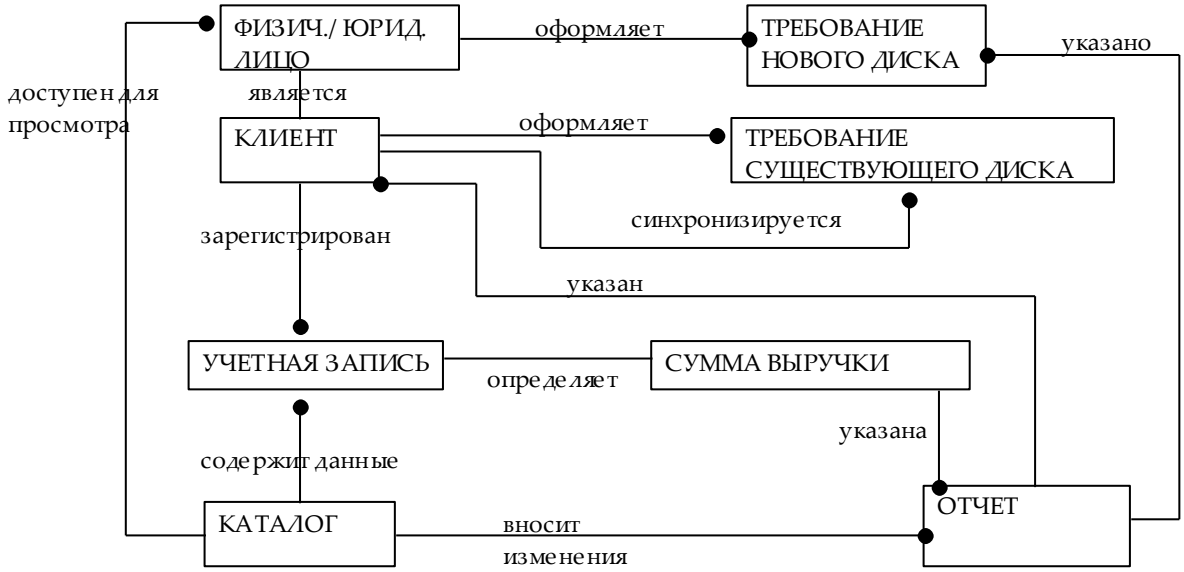
Концептуальная модель данных построена с использованием методологии информационного моделирования IDEF1X (Integrated DEFinitions 1 eXpanded), имеющей в настоящее время статус федерального стандарта США. В IDEF1X различают три уровня графического представления информации.

Уровень «сущность — связь» (***Entity-Relationship***, ***ER level***). Это уровень наименее детального представления информации. Сущности и связи представлены только именами.

Уровень ключей (***Key-Based***, ***KB level***). На этом уровне в диаграммах отражаются имена первичных и внешних ключей сущностей и связей. Диаграмма KB–уровня объявляет уникальные идентификаторы экземпляров сущностей и ограничения ссылочной целостности.

Уровень атрибутов (***Fully attributed***, ***FA level***). Диаграмма FA-уровня показывает имена всех атрибутов сущностей и связей и полностью определяет структуру и взаимосвязи данных. Включает в себя KB-уровень.

3.1 Сущности и связи (*ER-уровень*)



3.2 Проектирование реляционной модели данных на основе принципов нормализации

Цель нормализации — преобразовать универсальное отношение, представленное выше диаграммой «сущность-связь», в систему отношений, удовлетворяющих определенным формальным отношениям. При обновлении данных, хранящихся в такой системе отношений, не возникают аномальные ситуации, и РСУБД в состоянии поддерживать целостность данных.

Первая нормальная форма (1***НФ***)**.** Говорят, что отношение находится в первой нормальной форме, если и только если все домены его атрибутов содержат скалярные значения.

Любое отношение реляционной модели данных (РМД) находится в 1НФ по определению.

Вторая нормальная форма (1***НФ***)**.** Говорят, что отношение находится во второй нормальной форме, если и только если каждый его неключевой атрибут неприводимо зависит от первичного ключа.

Для приведения отношения ко второй нормальной форме необходимо выделить в отдельные отношения группы атрибутов, зависящих от части возможного ключа отношения 1НФ.

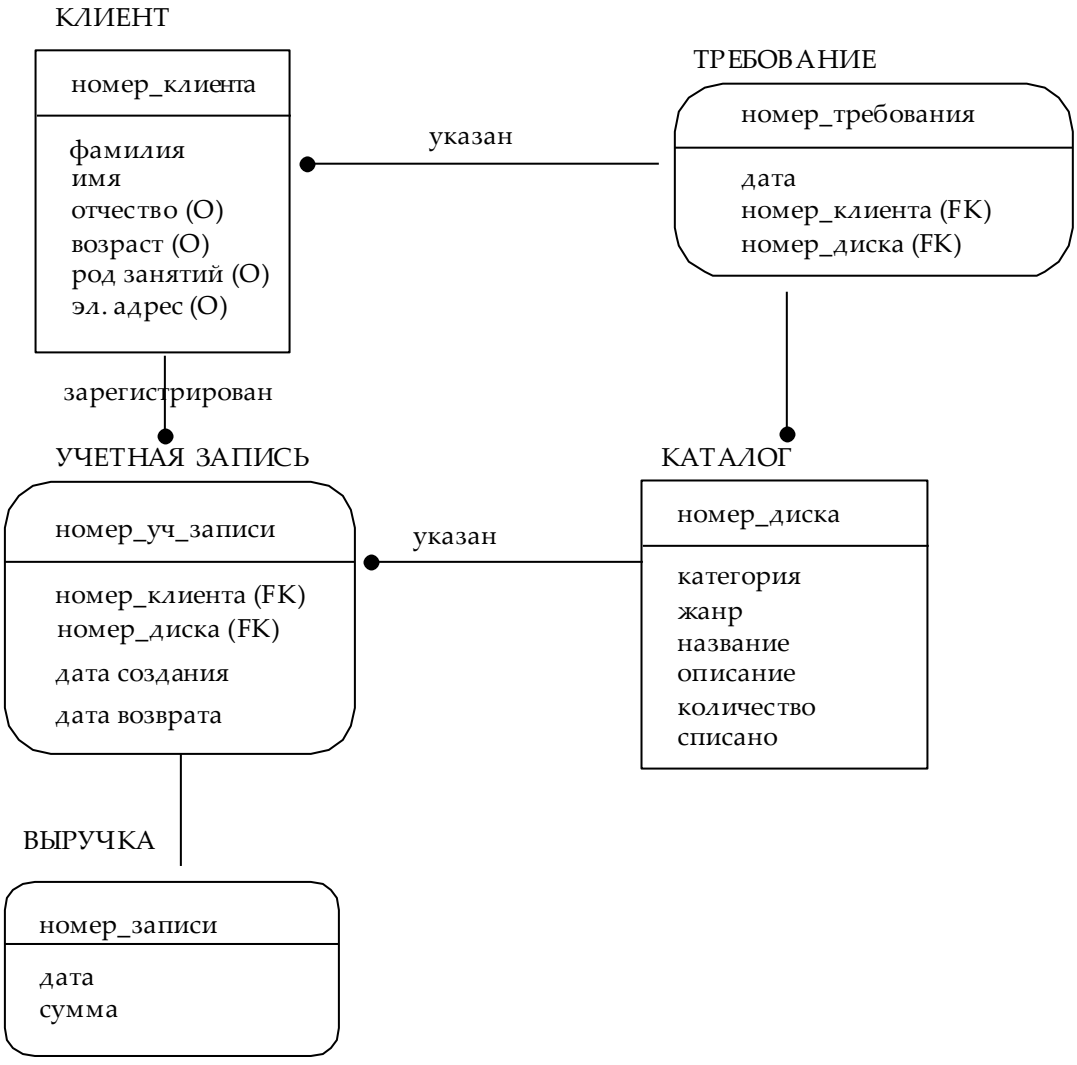
При несоблюдении этого условия отношения ТРЕБОВАНИЯ и УЧЕТНАЯ ЗАПИСЬ были бы объединены.

Третья нормальная форма (3***НФ***)**.** Говорят, что отношение находится в третьей нормальной форме, если и только если оно находится в 2НФ и нет транзитивных зависимостей, то есть все неключевые атрибуты взаимно независимы.

В данном случае этому условию не удовлетворяет отношение КАТАЛОГ, так как записи содержат названия дисков, зависящие от атрибутов *категория* и *жанр*.

Но при разбиении отношения на два отношения *категория — жанр* и *жанр — название* усложнилась бы работа с БД в прикладной программе.

3.3 Состав атрибутов сущностей (***FA-уровень***)



3.4 Глоссарий

Таблица 1 — Сопоставление физических и логических имен модели

Физическое имя	Логическое имя	Тип поля	Описание
Client_id	Номер клиента	autoincre-ment	Уникальный идентифика-тор клиента
Surname	Фамилия	Char(15)	Фамилия клиента
Name	Имя	Char(10)	Имя клиента
Patronymic	Отчество	Char(15)	Отчество. Необязательное поле
Age	Возраст	Numeric	Возраст. Необязательное поле
Job	Род занятий	Char(25)	Род занятий. Необяза-тельное поле
E-mail	Электронный адрес	Char(20)	Электронный адрес кли-ента. Необязательное по-ле.
Disk_id	Номер диска	autoincre-ment	Уникальный идентифика-тор диска. Владелец – КАТАЛОГ
Description	Описание	Char(15)	Ссылка на htm-файл с описанием диска. Владе-лец — КАТАЛОГ
Category	Категория	Char(30)	Категория. Используется для поиска нужного диска
Genre	Жанр	Char(30)	Жанр. Используется для поиска нужного диска
Caption	Название	Char(50)	Название диска. Владелец — КАТАЛОГ
Quantity	Количество	numeric	Количество экземпляров диска в КАТАЛОГе
Write off	Списано	numeric	Количество списанных дисков
Date	Дата	Date	Дата создания учетной за-писи, либо фактическая дата возврата диска, ис-пользуемая для вычисле-ния сдачи

Окончание табл. 1

Физическое имя	Логическое имя	Тип поля	Описание
ReturnDate	Дата возврата	Date	Формально указываемая дата возврата диска
Record_id	Номер уч. Записи	autoincrement	Уникальный идентификатор учетной записи
Req_id	Номер требуемого диска	autoincrement	Уникальный идентификатор записи об интересующем клиента диске
ReqDisk	Название требуемого диска	Char(30)	Название требуемого диска
Summa	сумма	Long Integer	Сумма, полученная после каждой операции продажи или проката диска

4. ОБОСНОВАНИЕ ВЫБОРА ПРОГРАММНЫХ СРЕДСТВ

В качестве среды разработки выбрана система быстрой разработки (RAD, Rapid Application Development) Delphi 7 как наиболее полно поддерживающая технологии визуального проектирования и методологии объектно-ориентированного событийного программирования.

В настоящее время существует достаточно большое количество программных систем, позволяющих создавать и использовать локальные (dBASE, FoxPro, Access, Paradox) и удаленные (Interbase, Oracle, Sysbase, Informix, Microsoft SQL Server) базы данных.

В состав Delphi 7 входят компоненты, позволяющие писать программы работы с файлами данных различных систем: от dBASE до Informix и Oracle. Кроме того, Delphi предоставляет утилиту Borland Database Desktop, позволяющую программисту создавать файлы баз данных в различных форматах.

Возможности по созданию приложений различного назначения, в том числе приложений БД, превосходят возможности специализированных СУБД.

ПРИЛОЖЕНИЕ Б

Задание на курсовую работу

Федеральное агентство по образованию
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

УТВЕРЖДАЮ
Заведующий кафедрой КИБЭВС,
д-р техн. наук, профессор
_____ А.А. Шелупанов
«___» _____ 20__ г.

ЗАДАНИЕ

на курсовую работу
студенту _____группы _____
факультета _____.

1 Тема работы: Проектирование учебно-исследовательской базы данных:
_____

2 Исходные данные к работе:
2.1 Реляционная СУБД _____
2.2 Данные по предметной области и атрибуты объектов.
3 Содержание курсовой работы:
3.1 Проектирование инфологической модели данных:
– структуризация предметной области;
– представление модели «Сущность-связь»;
– сценарий пользовательского интерфейса.
3.2 Проектирование датологической модели данных:
– проектирование структуры базы данных;
– написание программы обработки и работы с данными:

а) генерация программы меню, реализующей пользовательский интерфейс;

б) использование режимов редактирования данных;

в) режим просмотра данных с использованием сгенерированной экранной формы;

г) процедуры поиска и манипулирования данными;

д) возможность получить отчетную форму на принтер и экран с максимальным использованием средств генератора отчетов.

4 Содержание пояснительной записки:

– задание;
– постановка задачи;
– реферат;
– введение;
– вопросы проектирования БД;
– обоснование выбора программных средств;
– описание прикладной программы;
– руководство пользователю;
– заключение;
– список использованных источников;
– приложения (структуры БД, экранные формы, отчетная форма, листинг программы).

5 Дата выдачи задания: _____

6 Срок сдачи готовой работы: _____

Задание согласовано:
Руководитель работы

«___» _____ 200_ г. _____

Задание принято к исполнению
«___» _____ 200_ г. _____ _____