МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
АВТОМОБИЛЬНО-ДОРОЖНЫЙ ИНСТИТУТ

Н.В. Гуменюк

БАЗЫ ДАННЫХ

Учебное пособие
для обучающихся образовательных учреждений
высшего профессионального образования

Донецк
2020

УДК 004.65 : 658 (075.8)
ББК У212.5
Г945

Рекомендовано Ученым советом
ГОУВПО «Донецкий национальный технический университет»
в качестве учебного пособия для обучающихся образовательных учреждений
высшего профессионального образования
(Протокол № 1 от 28.02.2020)

Рецензенты:
Лепа Роман Николаевич – доктор экономических наук, профессор, заведующий отделом моделирования экономических систем ГУ «Институт экономических исследований»;
Моисеенко Игорь Алексеевич – доктор физико-математических наук, доцент, декан факультета математики и информационных технологий ГОУВПО «Донецкий национальный университет».

Автор:
Гуменюк Наталья Владимировна – доцент кафедры математического моделирования АДИ ГОУВПО «ДОННТУ»;

Гуменюк, Н. В.
Г945 Базы данных : учеб. пособие для обучающихся образоват. учреждений высш. проф. образования / Н. В. Гуменюк ; ГОУВПО «ДОННТУ». – Донецк : ДОННТУ, 2020. – 205 с. : ил., табл.

Учебное пособие содержит сведения по основным разделам дисциплины «Базы данных». В рамках теоретического курса изложены основополагающие принципы разработки современных баз данных, проанализированы модели данных, особое внимание уделено этапам концептуального и логического проектирования. Представленные в пособии практические рекомендации позволяют приобрести навыки работы с элементами реляционной базы данных MS Access, и развить умения использования структурированного языка запросов для манипулирования данными базы. Пособие содержит примеры и методики выполнения типовых заданий для самостоятельной проработки, а также перечень контрольных вопросов, позволяющий оценить уровень усвоения материала.

Книга предназначена для обучающихся вузов, преподавателей и специалистов в области проектирования баз данных.

УДК 004.65 :658 (075.8)
ББК У212.5

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящее время характеризуется небывалым ростом объема информационных потоков. Это относится практически к любой сфере деятельности человека. Наибольший рост объема информации наблюдается в промышленности, торговле, финансово-банковской и образовательной сферах. Для оперативного учета и накопления возрастающих объемов информации соответствующей предметной области, ее структуризации, обработки и представления в удобном для пользователей виде с целью повышения эффективности принимаемых управленческих решений используются информационные системы, ядро которых составляют технологии баз и банков данных. В самом общем виде можно сказать, что базы данных (БД) – это большие массивы информации о какой-либо сфере производственной или общественной деятельности, предназначенные для коллективного использования и допускающие компьютерную обработку этой информации.

Умение грамотно работать с современными базами данных является одним из ключевых требований к любому специалисту в области компьютерных технологий.

Важную роль в освоении технологий баз данных играет понимание теоретических основ их проектирования. Предлагаемое учебное пособие нацелено на ознакомление студентов с фундаментальными понятиями баз данных. В нем дается общий обзор видов баз данных, моделей и технологий многопользовательских систем управления базами данных (СУБД). Особое внимание уделяется наиболее распространенным реляционным базам данных и рассмотрению ключевых этапов их проектирования. Описываются основные конструкции языка SQL, являющегося фактическим стандартом работы с реляционными базами данных.

Для закрепления теоретических знаний на практике в пособии рассмотрен сквозной пример работы с СУБД Access, компании Microsoft. С помощью Access

можно разрабатывать базы данных различной сложности: от индивидуальной, предназначенной для использования на отдельном компьютере, до систем масштаба предприятия, работающих в компьютерных сетях.

Поэтапное изучение материала и выполнение практических заданий позволяет выработать навыки создания и умения работы с ключевыми компонентами базы данных: таблицами, формами, отчетами, запросами, макросами.

Таким образом, целью изучения технологии баз данных является формирование базы теоретических знаний и практических навыков, являющихся основой для успешного осуществления практической деятельности, связанной с проектированием, реализацией и управлением информационными системами и базами данных.

В результате изучения представленного материала пособия студент должен знать:

– принципы организации современных БД и СУБД;

– основные инфологические и даталогические модели данных;

– теоретические и математические основы реляционной модели данных;

– реляционную алгебру; современные языки манипулирования данными в реляционной модели QBE и SQL;

– основы физической организации БД;

– основные методы защиты информации, применяемые в базах данных.

Уметь:

– корректно проектировать реляционные базы данных с учетом функциональных зависимостей;

– применять на практике теорию нормализации; составлять запросы к базе данных произвольной сложности на языке реляционной алгебры, QBE и SQL;

– применять на практике технологию БД для разработки конкретных систем; работать в СУБД типа MS Access.

Владеть навыками работы с СУБД для создания собственных БД, выполнения операций по корректировке, выборке и поиску информации в базе.

ТЕМА 1 КЛЮЧЕВЫЕ ПОНЯТИЯ ТЕОРИИ БАЗ ДАННЫХ

1.1 Информация, данные, знания: различные аспекты трактовки

К базовым понятиям, которые используются в информатике, относятся *данные, информация и знания*. Эти понятия часто используются как синонимы, однако между ними существуют принципиальные различия.

Термин «данные» происходит от слова «data» – факт, а понятие «информация» («informatiq») означает разъяснение, изложение, т.е. сведения или сообщение.

Информация – это любые сведения о каком-либо объекте, событии или процессе, над которыми выполняются действия: восприятия, передачи, преобразования, хранения и использования.

Информация может быть получена в результате наблюдений, измерений. Информация используется во всех областях человеческой деятельности, человек научился собирать, обрабатывать информацию, передавать ее по назначению. С понятием информации неразрывно связано понятие данные.

Данные – информация, представленная в формализованном виде, пригодном для передачи, интерпретации или обработки с участием человека или автоматическими средствами.

Системы, служащие для регистрации, обработки, хранения и передачи информации, называют **информационными системами** (ИС). ИС, являясь системой по сбору, передаче и обработке информации об объекте, обеспечивает сотрудников некоторой организации информацией, достаточной для реализации функций управления.

Итак, в базах данных хранятся различные данные, а по определенному запросу система управления базой данных выдает требуемую информацию.

Основное отличие информации заключается в том, что в отличие от данных она всегда имеет смысловую нагрузку

Соответственно двум понятиям — «информация» и «данные» – при проектировании баз данных различают два аспекта рассмотрения вопросов: инфологический и датологический [1].

Инфологический аспект употребляется при рассмотрении вопросов, связанных со смысловым содержанием данных независимо от способов их представления в памяти системы. На этапе инфологического проектирования информационной системы должны быть решены вопросы:

– о каких объектах или явлениях реального мира требуется накапливать и обрабатывать информацию в системе;

– какие их основные характеристики и взаимосвязи между собой будут учитываться;

– уточнения вводимых в информационную систему понятий об объектах и явлениях, их характеристиках и взаимосвязях.

Таким образом, на этапе инфологического проектирования выделяется часть реального мира, определяющая информационные потребности системы, т. е. ее **предметную область.**

Даталогический аспект употребляется при рассмотрении вопросов представления данных в памяти информационной системы. При даталогическом проектировании разрабатываются соответствующие формы представления информации в системе посредством данных, а также приводятся модели и методы представления и преобразования данных, формулируются правила смысловой интерпретации данных, т.е. формируется **семантика данных**. Данные, выражающие семантику данных, называются **метаданными.**

Работа с семантикой – это работа со **знаниями**. Основное средство представления семантики данных – естественный язык. Но можно использовать формализованные языки, которые позволяют более эффективно организовать обработку данных на вычислительной технике и представить необходимую

семантику данных, удовлетворяющую практическим потребностям целого ряда прикладных задач. К этому классу ИС относятся и банки данных

Знания – это зафиксированная и проверенная практикой обработанная информация, которая использовалась и может многократно использоваться для принятия решений. С другой стороны, знания – это вид информации, которая хранится в базе знаний и отображает знания специалиста в конкретной предметной области. Знания – это интеллектуальный капитал.

Формальные знания могут быть в виде документов (стандартов, нормативов), регламентирующих принятие решений, или учебников, инструкций с описанием решения задач.

Неформальные знания – это компетенция и опыт специалистов в определенной предметной области.

1.2 Понятие банка и базы данных

В основе концепции баз данных лежит механизм предоставления обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе.

Под **базой данных** (БД) понимается именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области. Характерной чертой баз данных является постоянство: данные постоянно накапливаются и используются; состав и структура данных, необходимых для решения тех или иных прикладных задач, обычно постоянны и стабильны во времени; отдельные или даже все элементы данных могут меняться, но и это есть проявление постоянства – постоянная актуальность.

БД и СУБД являются составными частями более сложной системы, именуемой банком данных [2]

Банк данных (БнД) – это система специально организованных данных, программных, языковых, технических, организационно-методических средств, предназначенных для централизованного накопления и коллективного

многоцелевого использования данных. Следует отметить, что термин «банк данных» используется сравнительно редко. В современной литературе понятие «банк данных» ассоциируется с системами баз данных.

Система баз данных представляет собой совокупность программного обеспечения, данных и аппаратного обеспечения компьютеров, которая реализует набор приложений и моделей данных, и использует СУБД и прикладное программное обеспечение для создания конкретной информационной системы [3].

Основными требованиями, предъявляемыми к БнД, являются [4]:

– адекватность отображения предметной области (полнота, целостность и непротиворечивость данных, актуальность информации (т. е. ее соответствие состоянию объекта на данный момент времени));

– возможность взаимодействия пользователей разных категорий и в разных режимах, обеспечение высокой эффективности доступа для разных приложений;

– дружелюбность интерфейсов и быстрое освоение системы, особенно для конечных пользователей;

– обеспечение секретности и конфиденциальности для некоторой части данных;

– определение групп пользователей и их полномочий;

– обеспечение взаимной независимости программ и данных;

– обеспечение надежности функционирования БнД, защита данных от случайного и преднамеренного разрушения;

– возможность быстрого и полного восстановления данных в случае их разрушения;

– технологичность обработки данных, приемлемые характеристики функционирования БнД (стоимость обработки, время реакции системы на запросы, требуемые машинные ресурсы и др.).

БнД является сложной человеко-машинной системой, состоящей из взаимозависимых компонентов (рис. 1).

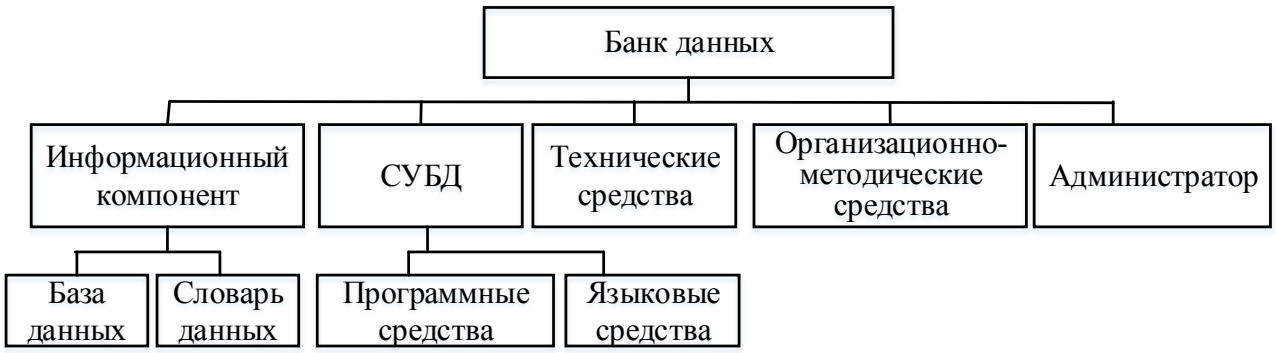


Рис. 1 – Компоненты банка данных

Информационный компонент БнД содержит базу данных и словарь данных. Словарь данных является хранилищем метаинформации, т.е. информации о информации. Метаинформация включает в себя описание базы данных, информацию о предметной области, представленной в БД, сведения о пользователях БнД и т.д.

Система управления базами данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Иными словами, СУБД играет роль интерфейса между прикладными программами и базой данных.

СУБД предоставляет пользователю программные и языковые средства, обеспечивающие взаимодействие всех частей информационной системы при ее функционировании.

Программные средства БнД обеспечивают функционирование СУБД. Здесь можно выделить ядро СУБД, обеспечивающее организацию ввода, обработки и хранения данных, а также другие компоненты, обеспечивающие настройку системы, средства тестирования, утилиты, обеспечивающие выполнение вспомогательных функций (восстановление баз данных, сбор статистики о функционировании БнД и др.). Важной компонентой СУБД являются трансляторы или компиляторы для используемых ею языковых средств.

Языковые средства СУБД относятся к языкам четвертого поколения. Они предназначаются для пользователей разных категорий: конечных пользователей, системных аналитиков, профессиональных программистов. В любой базе обязательно есть языки двух типов: языки описания данных и языки манипулирования данными.

Технические средства, используемые в БнД, – это компьютеры разных классов, периферийные устройства, коммуникационная (сетевая) аппаратура.

Организационно-методические средства представляют собой различные инструкции, методические и регламентирующие материалы, предназначенные для пользователей, взаимодействующих с БнД.

Администраторы – это специалисты, которые обеспечивают создание, функционирование и развитие БнД.

База данных (database) – это организованная совокупность совместно используемых логически связанных данных и описаний этих данных, относящаяся к определенной предметной области, предназначенная для удовлетворения информационных потребностей организации.

Дополним данное определение БД рядом комментариев.

1. База данных представляет собой долговременное хранилище данных, структура которого определяется на этапе проектирования БД.

2. Находящиеся в БД данные являются общекорпоративным ресурсом, в котором вместо отдельных файлов, разбросанных по отделам и службам предприятия, данные собраны вместе с минимальной избыточностью.

3. В БД хранятся не только рабочие данные, описывающие какую-то предметную область, но и метаданные, предназначенные для БД часто называют самодокументированием.

4. Потребителем данных может выступать не только человек, но и программно-аппаратные комплексы (станки с числовым программным управлением, автоматизированные системы управления, робототехнические системы и т. д.), именно поэтому говорится об удовлетворении информационных потребностей не просто пользователей, а организации в целом [5].

1.3 Персонал и пользователи базы данных

Совместное использование данных на предприятии, с одной стороны, предоставляет сотрудникам уникальную возможность всегда находиться в курсе текущего состояния дел, а с другой – порождает неминуемые конфликты интересов за право владения и обработки данных. Необходимость поддержки на предприятии идеологии совместной работы с данными привела к появлению специальных отделов, отвечающих за информатизацию, в сферу интересов которых должны входить следующие группы функций:

1) ***сервисные функции***, направленные на поддержку сотрудников предприятия (конечных пользователей) в области доступа к данным;

2) ***производственные функции*** по удовлетворению специфичных информационных потребностей сотрудников предприятия в соответствии с решаемыми ими задачами;

3) ***функции обеспечения информационной безопасности***, в первую очередь направленные на запрет доступа к конфиденциальной информации несанкционированных лиц.

Сотрудники отдела информатизации, на которых возложено выполнение перечисленных функций, вместе с работниками других отделов и служб, использующих приложения для решения типовых задач производственно-хозяйственной деятельности, составляют круг пользователей БД:

–администратор данных, АД (Data Administrator, DA);

– администратор(ы) БД, АБД (Database Administrator, DBA);

– разработчики БД;

– прикладные программисты;

– конечные пользователи.

Представленная на рисунке 2 схема отображает основные направления деятельности персонала предприятия при работе с БД.

Администратор данных

С точки зрения управленческих функций в отделе информатизации на вершине пирамиды располагается администратор данных. Это наиболее опытный специалист, в первую очередь выполняющий организаторские функции, кроме того, АД отвечает за управление данными (планирование БД, разработка стандартов, бизнес-правил) и за концептуальное проектирование базы данных.

Наиболее значимые **задачи администратора данных** [6-8]**:**

– оказание помощи в формировании корпоративной стратегии построения информационной системы (ИС);

– контроль за внедрением и неукоснительным исполнением на всех этапах жизненного цикла ИС *политик, процедур и стандартов* компании по корректному созданию, использованию и распространению данных;

– разработка концептуальной модели данных;

– планирование процесса создания БД;

– выявление требований предприятия к используемым данным;

– установка правил сбора, хранения и представления данных;

– определение политики информационной безопасности;

– разработка концептуальной и логической моделей БД;

– взаимодействие с АБД и программистами с целью обеспечения соответствия разрабатываемой БД и клиентских приложений существующим стандартам и требованиям предприятия;

– контроль за постоянной модернизацией ИС и ПО;

– обеспечение полноты требуемой документации;

– взаимодействие с конечными пользователями ИС.

Для небольших предприятий вполне допустимо, чтобы обязанности АД совмещал начальник отдела информатизации или АД выступал внештатным сотрудником, активно принимающим участие в разработке либо глубокой модернизации информационной системы.

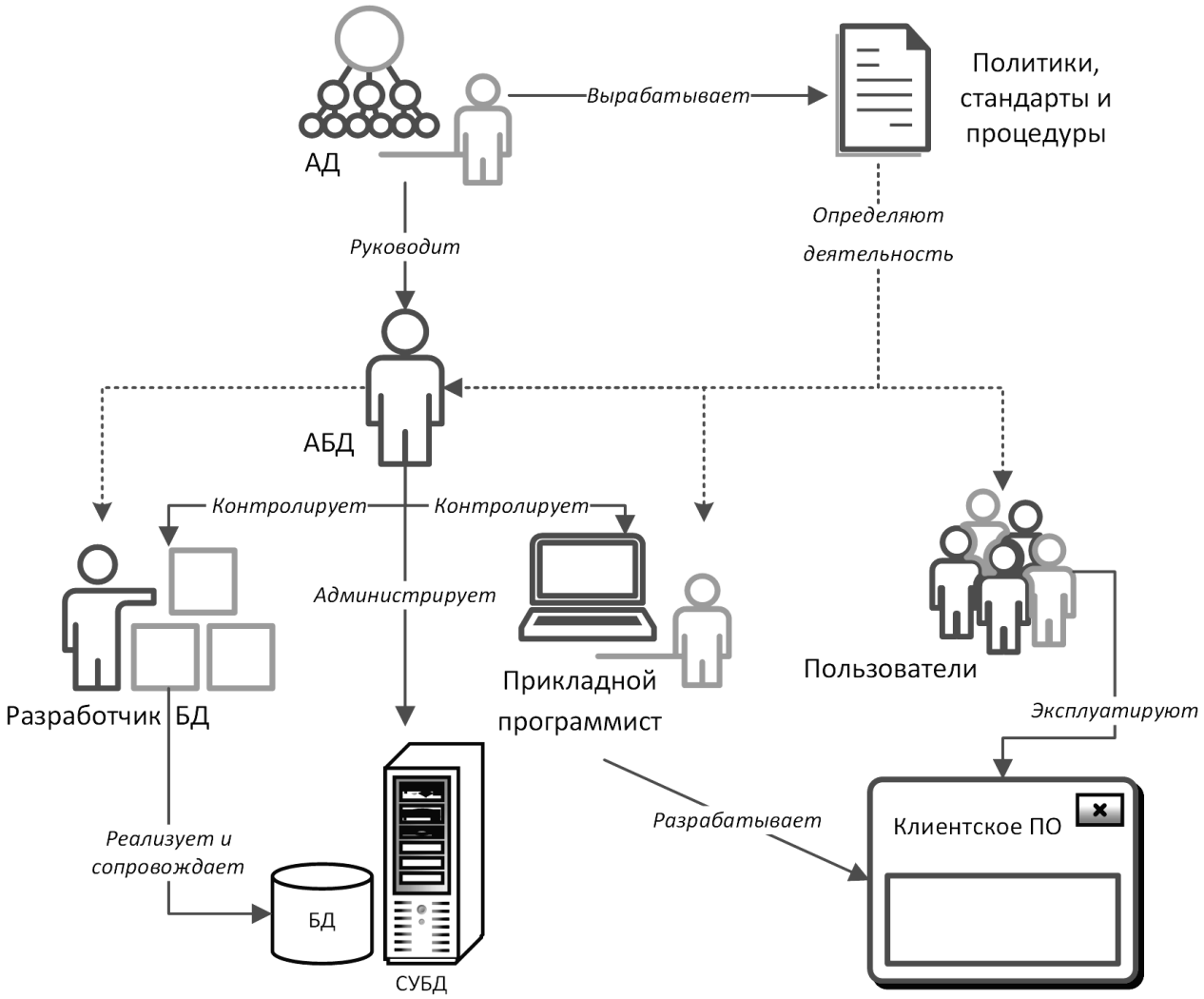


Рис. 2 – Общее направление деятельности персонала при работе с БД

Под ***политикой (policies)*** понимается ключевая руководящая инструкция, подлежащая неукоснительному соблюдению персоналом.

Например, в период проектирования АД вводит политику именования объектов БД (таблиц, столбцов, индексов, хранимых процедур и т. д.), которой должны придерживаться АБД и программисты. Другим примером политики, на этот раз на этапе эксплуатации БД, станет требование, чтобы все пользователи могли работать с данными исключительно на основе парольного доступа.

Стандарты выступают развитием политик, они формируют их более детализированное описание. Допустим, стандарты укажут, как следует

именовать первичные ключи таблиц или минимальную длину пароля пользователя.

Процедуры представляют собой тексты инструкций, содержащие последовательность действий, которые необходимо выполнить персоналу в том или ином случае работы с БД.

Администратор(ы) базы данных отвечает за реализацию базы данных (в том числе за физическое проектирование), за обеспечение безопасности и целостности данных, за сопровождение системы, а также за обеспечение максимальной производительности СУБД и приложений.

К **задачам администратора баз данных** относятся:

– выбор целевой СУБД;

– разработка логической модели БД;

– обеспечение требуемой защиты и целостности данных;

– тестирование БД;

– введение БД в эксплуатацию;

– подготовка пользователей к работе с БД;

– системное администрирование СУБД (контроль производительности, резервное копирование, восстановление, репликация и т. д.);

– ведение технической документации БД;

– сопровождение БД.

Разработчики баз данных

Нуждающиеся в проектировании и последующем сопровождении больших БД крупные компании в штатный состав отделов информатизации могут вводить разработчиков БД. На более мелких предприятиях эти функции возлагаются на АБД или делегируются компании, у которой приобретается программное обеспечение.

Разработчики БД, руководствуясь концептуальной моделью данных и принятыми на предприятии политиками и стандартами, осуществляют разработку логической, а затем и физической моделей данных.

Прикладные программисты

В современных клиент-серверных системах БД развертываются на стороне сервера. Для того чтобы услугами БД смогли воспользоваться обычные сотрудники предприятия, им, как правило, прямой доступ к серверу не предоставляется, на высокоуровневых языках программирования создается отдельное программное обеспечение – клиентские приложения.

Разработка клиентского ПО относится к зоне ответственности прикладных программистов. В их **задачи** входит:

– организация доступа клиентского приложения к БД;

– проектирование интерфейса пользователя;

– наполнение приложения заданным функционалом;

– тестирование и отладка приложений;

– сопровождение приложений в период их эксплуатации.

Конечные пользователи

Главным потребителем услуг информационной системы предприятия выступают конечные пользователи. Их можно классифицировать по следующим параметрам:

– по уровню управления (высшее руководство предприятия, менеджмент среднего звена, рядовые сотрудники);

– по уровню знаний (опытный пользователь, обычный пользователь, начинающий пользователь);

– по частоте обращения к данным (редко, периодически, часто).

Независимо от того, существует ли на предприятии отдел информатизации, либо его функции возложены на одного сотрудника (в случае, если речь идет о малом предприятии) вся деятельность относительно работы с БД должна быть нацелена на обеспечение доступа конечных пользователей к информационным ресурсам организации. Поэтому АД и АБД, обязаны на постоянной основе:

– осуществлять сбор и анализ требований пользователей к ИС предприятия с целью ее развития и совершенствования;

– поддерживать целостность данных в БД, в том числе путем выявления и предотвращения некорректных действий пользователей;

– своевременно устранять проблемы доступа пользователей к данным;

– обеспечивать безопасность данных, обрабатываемых пользователями;

– осуществлять обучение и поддержку пользователей.

При этом должен соблюдаться главный принцип: от конечных пользователей не должно требоваться каких-либо специальных знаний в области вычислительной техники и языковых средств. Соответственно, проектируемая БД, с одной стороны, должна обладать максимально дружественным интерфейсом, а с другой – уметь противостоять не только действиям разнообразных хакеров, но и адекватно реагировать на некорректные операции легальных пользователей системы.

1.4 Классификация баз данных

По форме представления информации различают: видео-и аудиосистемы, а также системы мультимедиа. Эта классификация в основном показывает, в каком виде информация из баз данных выдается пользователям: в виде изображения, звука или в виде сочетания разных форм отображения информации.

Наиболее распространенным являются базы данных, содержащих обычные *символьные данные*. Эти БД подразделяются:

– *неструктурированные БД* – базы, организованные в виде семантических сетей;

– *частично структурированными* можно считать БД, в которых информация представлена в виде обычного текста или гипертекста;

– *структурированные БД.*

Структурированные БД ***по типу используемой модели*** делятся на:

– иерархические;

– сетевые;

– реляционные;

– объектно-ориентированные;

– смешанные;

– мультимодельные.

Наибольшее коммерческое использование в настоящее время имеют *реляционные БД.*

По типу хранимой информации БД делятся на:

– документальные БД содержат сведения о документах на естественном языке – монографиях, научных отчетах, текстах законодательных актов и т.д.;

– фактографические БД содержат фактические сведения, например, данные о кадровом составе предприятия.

По характеру организации хранения данных и обращения к ним:

– локальные (персональные),

– общие (интегрированные);

– распределенные БД.

По условиям предоставления услуг различают:

– бесплатные БД;

– платные БД. Платные БД делятся на бесприбыльные и коммерческие.

По форме собственности БД делятся на:

– государственные;

– негосударственные.

По степени доступности различают:

– общедоступные;

– с ограниченным кругом пользователей.

По количеству пользователей выделяют:

– персональные системы – предназначены для создания небольших БД, устанавливаемых на одном компьютере, поэтому их часто называют настольными;

– многопользовательские системы – предназначены для обслуживания БД, находящихся в совместном владении несколькими пользователями.

1.5 Архитектура баз данных

Построению систем, основанных на использовании СУБД, предшествовали интенсивные научные исследования, направленные на решение проблемы устройства самой СУБД. Наиболее фундаментальным результатом этих исследований стало определение трех уровней абстракции (рис. 3), то есть трех различных уровней описания элементов данных, зафиксированных в модели ANSI-SPARC [6].

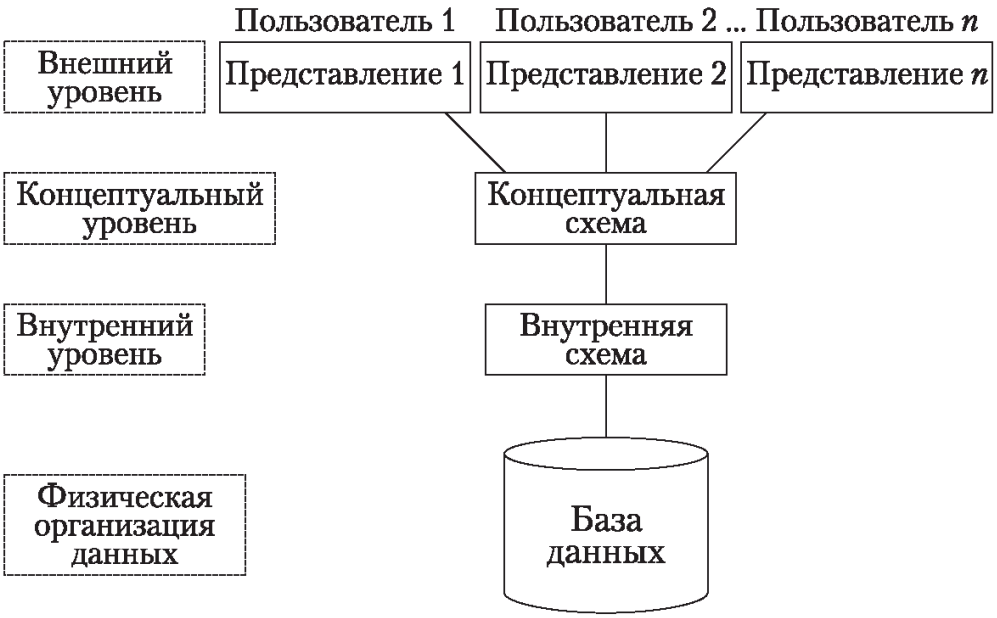


Рис. 3 – Трехуровневая архитектура ANSI-SPARC

Эти уровни формируют трехуровневую архитектуру базы данных, которая охватывает ***внешний, концептуальный и внутренний уровни.***

Уровень, на котором данные воспринимаются пользователями, называется внешним уровнем (external level), тогда как СУБД и операционная система воспринимают данные на внутреннем уровне (internal level). Именно на внутреннем уровне данные реально сохраняются с использованием структур данных и файловой организации. Концептуальный уровень (conceptual level)

представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости друг от друга.

Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Такое разделение базы данных на уровни объясняется следующими причинами:

– каждый пользователь должен иметь возможность обращаться к одним и тем же данным, реализуя свое собственное представление о них;

– каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей;

– пользователи не должны непосредственно иметь дело с какими-либо подробностями физического хранения данных в базе, т.е. взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных;

– администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления;

– внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переключение на новое устройство хранения;

– администратор базы данных должен иметь возможность изменять концептуальную структуру базы данных без какого-либо влияния на всех пользователей.

Внешний уровень – это представление базы данных с точки зрения пользователей. Он описывает ту часть базы данных, которая относится к каждому пользователю. Внешний уровень состоит из нескольких различных внешних представлений базы данных. Каждый пользователь имеет дело с представлением «реального мира», выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи

«реального мира», которые интересны пользователю. Другие сущности, атрибуты или связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не подозревать об их существовании.

Концептуальный уровень соответствует обобщающему представлению базы данных. Этот уровень описывает то, какие данные хранятся в базе данных, а также связи, существующие между ними. Он содержит логическую структуру всей базы данных (с точки зрения администратора базы данных). Фактически это полное представление требований к данным со стороны предприятия, которое не зависит от способа их хранения. На концептуальном уровне представлены следующие компоненты:

– все сущности, их атрибуты и связи;

– накладываемые на данные ограничения;

– информация о смысловом содержании данных;

– информация о мерах обеспечения безопасности и поддержки целостности данных.

Внутренний уровень отражает физическое представление базы данных в компьютере, описывая, как информация хранится в базе данных. Он описывает физическую реализацию базы данных и предназначен для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. Он содержит описание структур данных и организации отдельных файлов, используемых для хранения данных на запоминающих устройствах. На этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы (вспомогательными функциями хранения и извлечения записей данных) с целью размещения данных на запоминающих устройствах, создания индексов, извлечения данных и т.д. На внутреннем уровне хранится следующая информация:

– распределение дискового пространства для хранения данных и индексов;

– описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);

– сведения о размещении записей;

– сведения о сжатии данных и выбранных методах их шифрования.

Ниже внутреннего уровня находится ***физический уровень*** (physical level), который реализуется операционной системой, но под управлением СУБД. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут отличаться от системы к системе.

Трехуровневая архитектура баз данных отражается в **последовательности этапов их проектирования**. В базе данных отражается информация об определенной ***предметной области (ПО)***, которой называется часть реального мира, представляющая интерес для данного исследования или использования.

Чтобы спроектировать структуру базы данных, необходима исходная информация о предметной области. Желательно, чтобы эта информация была представлена в формализованном виде.

Описание предметной области, выполненное без ориентации на используемые в дальнейшем СУБД и технические средства, называется ***инфологической моделью (ИЛМ) предметной области.***

На основе ИЛМ строится ***даталогическая модель (ДЛМ) базы данных,*** которая представляет собой отображение логических связей между информационными элементами инфологической модели. Даталогическая модель строится в терминах информационных единиц, допустимых в той конкретной СУБД, в среде которой проектируется БД.

Этап создания ДЛМ называется ***даталогическим проектированием.***

Описание логической структуры БД на языке СУБД называется ***схемой.***

Для привязки ДЛМ к среде хранения используется ***модель данных физического уровня***, или физическая модель. Эта модель базы данных определяет используемые запоминающие устройства, способы физической организации данных в среде хранения.

Физическая модель так же, как и ДЛМ, строится с учетом возможностей, предоставляемых СУБД. Описание физической структуры БД называется

схемой хранения. Соответствующий этап проектирования БД называется ***физическим проектированием***.

В некоторых СУБД, помимо описания общей логической структуры БД (т.е. схемы), имеется возможность описать логическую структуру БД с точки зрения конкретного пользователя. Такая модель называется внешней, а ее описание называется ***подсхемой*** (рис. 4).

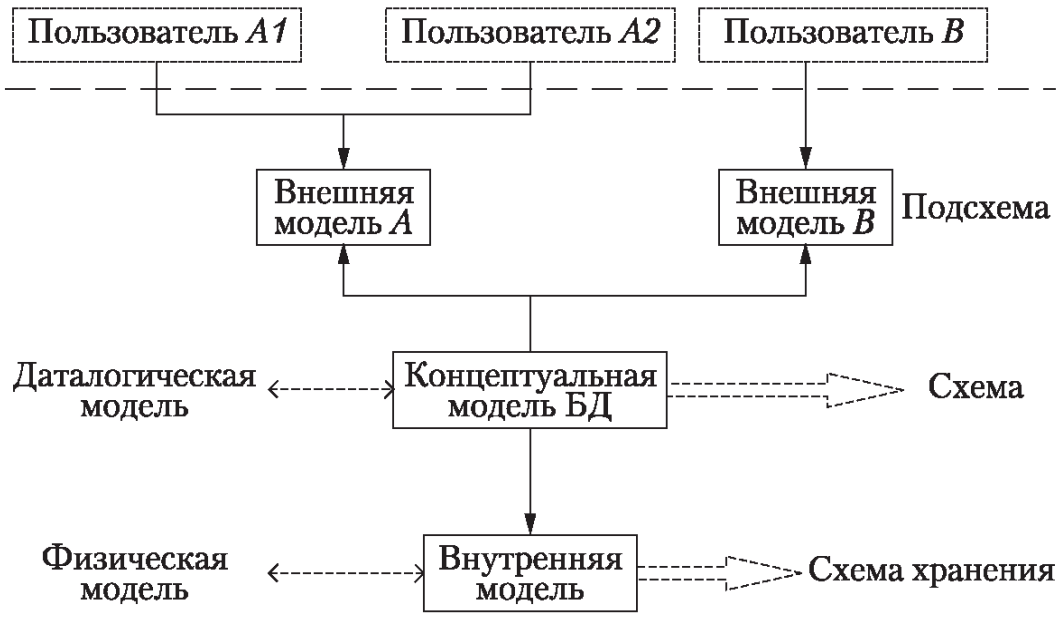


Рис. 4 – Модели и описания структуры БД, поддерживаемые СУБД

Если СУБД поддерживает уровень подсхем, то перед проектировщиком встает задача определения подсхем. Это также можно рассматривать как этап проектирования БД.

Использование подсхем облегчает работу пользователя, так как он должен знать структуру не всей базы данных, а только той ее части, которая имеет непосредственное отношение к нему; кроме того, эта структура приспособлена к его потребностям.

Взаимосвязь этапов проектирования базы данных показана на рисунке 5.

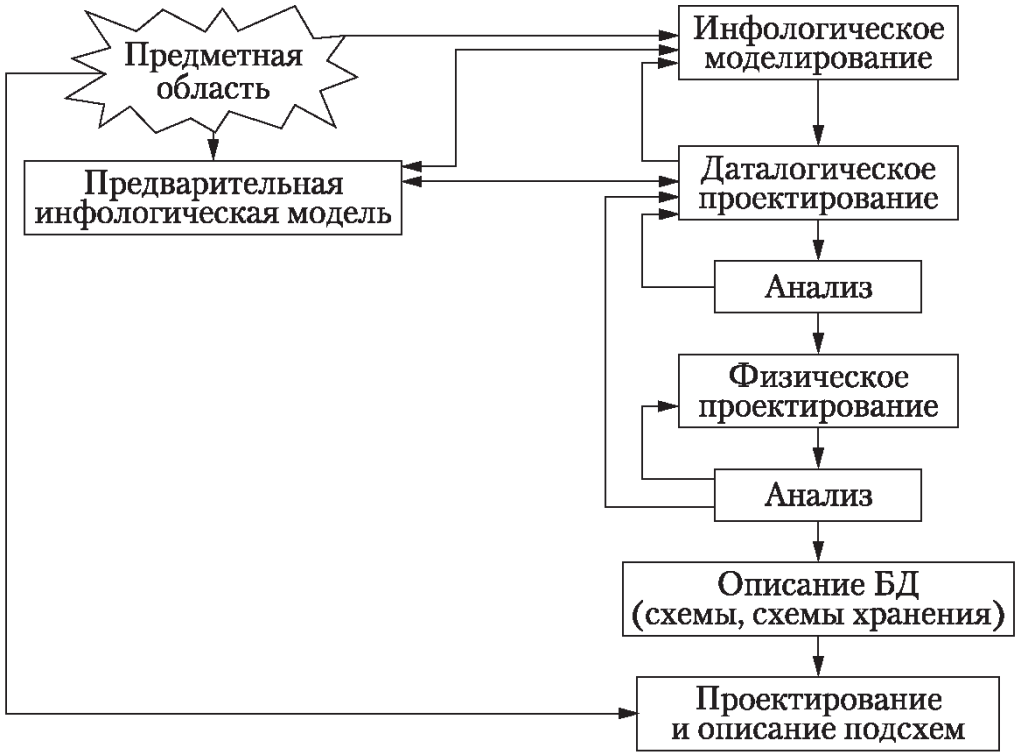


Рис. 5 – Взаимосвязь этапов проектирования

Первой строится инфологическая модель предметной области. Предварительная ИЛМ строится еще на предпроектной стадии и затем уточняется на более поздних стадиях проектирования.

Затем на ее основе строится даталогическая модель. Физическая и внешняя модели после этого могут строиться в любой последовательности, в том числе и параллельно. При проектировании БД возможен возврат на предыдущие этапы.

Возможны два вида возвратов: первый обусловлен необходимостью пересмотра результата проектирования (например, для улучшения полученных характеристик, «обхода» ограничений и т.п.), второй вызван необходимостью уточнения предыдущей модели (обычно инфологической) с целью поучения дополнительной информации для проектирования ли при выявлении противоречий в модели.

Контрольные вопросы:

1. В чем заключаются общие черты и в чем состоят различия в трактовке понятий «информация», «данные», «знания»?

2. Раскройте сущность инфологического и даталогического подходов к проектированию БД.

3. Проанализируйте роли составных элементов банка данных.

4. Дайте определение понятию «база дынных».

5. Какие функции возлагаются на информационный отдел предприятия?

6. Проанализируйте роли всех групп пользователей базы данных.

7. Перечислите классификационные признаки и соответствующие им типы баз данных.

8. Проанализируйте сущность трехуровневой архитектуры базы данных.

9. Аргументируйте связь трехуровневой архитектуры с этапами проектирования базы данных.

10. Проанализируйте взаимосвязь этапов проектирования базы данных.

ТЕМА 2 СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

2.1 Понятие и основной функционал СУБД

Система управления базами данных (Database Management System, DBMS) – это комплекс программных средств, с помощью которого можно создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ пользователей [5, 9].

Введение в определение СУБД понятия «контролируемый доступ» очень важно, т. к. акцентирует внимание на то, что на СУБД возлагаются обязанности не просто по управлению базами данных, но и по:

– предотвращению несанкционированного доступа к данным:

– контролю за многопользовательским (параллельным) доступом;

– поддержке целостности данных;

– восстановлению данных.

Функционал СУБД.

В самом общем случае работу СУБД можно описать следующим образом.

1. Пользователь запрашивает у системы разрешение на доступ к данным.

2. СУБД анализирует запрос и проверяет права пользователя на осуществление запрашиваемой операции.

3. СУБД выполняет требуемые действия с данными и при необходимости возвращает результат пользователю.

Ключевой функционал СУБД был сформулирован ***Эдгаром Фрэнком Коддом*** еще в начале 1980-х годов [10]. На первом этапе насчитывалось 8 функций, позднее Кодд и другие исследователи неоднократно расширяли и уточняли этот перечень. Рассмотрим основополагающие функции СУБД.

1. **Доступность данных**. Предоставление пользователю возможности вставлять, редактировать, удалять и извлекать данные из БД. При осуществлении любой из операций пользователь не должен вникать в

особенности физической реализации системы, т. е. все операции должны быть прозрачны.

2. **Метаописание данных**. СУБД должна предоставлять системный каталог, в котором содержится: описание хранимых в БД данных; описание связей между данными; ограничения целостности данных; регистрационные данные пользователей и другая служебная информация. Благодаря метаданным БД становится доступной внешним приложениям, упрощается понимание смысла данных, усиливаются меры безопасности, может выполняться аудит информации.

3. **Управление параллельностью**. Реализация механизма одновременного многопользовательского (параллельного) доступа к обрабатываемым данным с гарантией корректного обновления этих данных. Умение предоставить нескольким пользователям совместный доступ к разделяемым ресурсам – это едва ли не самая сложная задача, решаемая СУБД. СУБД должна суметь избежать конфликта совместного доступа двух или большего числа пользователей к одним и тем же строкам таблицы, или, по крайней мере, исключить какие-либо нежелательные последствия при возникновении конфликта.

4. **Обработка данных в рамках транзакции**. СУБД гарантирует что БД будет всегда находиться в непротиворечивом состоянии вне зависимости от любых сбоев при проведении операций обновления данных. Для этого операции с данными (в первую очередь вставки, редактирования и удаления) объединяются в единый блок, называемый транзакцией. Все операторы транзакции должны быть выполнены корректно и полностью, только в этом случае в БД будут зафиксированы изменения. В противном случае осуществляется автоматический откат транзакции, т. е. состояние БД будет восстановлено на момент времени, предшествующий вызову транзакции.

Например, Представьте себе, что вы снимаете какую-то сумму наличных денег в одном из банкоматов вашего города. Вы уже ввели свой код доступа, указали требуемую сумму, эта сумма денег списана с вашего электронного

счета и «бежит» по проводам из банка к вам в руки. И вдруг из-за сбоя питания, или отказа коммуникационного оборудования, или по какой-то другой технической причине команда на выдачу денег в банкомат не дошла. Что в результате? Программное обеспечение банковских платежных систем «увидит», что одна из операций транзакции завершена некорректно. Поэтому все изменения, сделанные указанной транзакцией, подлежат отмене – списанные электронные деньги вновь вернутся к вам на банковский счет. Вам остается повторить операцию получения наличных в другом банкомате.

5. **Обеспечение целостности данных.** Все содержащиеся в БД данные должны быть корректны и непротиворечивы. Это означает, что данные в таблицах могут модифицироваться только в соответствии с утвержденными правилами. В самом общем случае можно говорить о существовании трех правил поддержания целостности данных: целостность доменов, целостность отношений, целостность связей между отношениями. Кроме того, разработчик имеет возможность описывать свои собственные бизнес-правила, которые мы станем называть корпоративными ограничениями.

6. **Восстановление данных**. В случае непредвиденных ошибок и сбоев, приведших к повреждению или разрушению данных, СУБД должна обладать возможностью восстанавливать пострадавшие данные. В первую очередь эта функция реализуется с помощью процедур резервного копирования.

7. **Обмен данными.** СУБД обязана поддерживать современные сетевые технологии и предоставлять доступ к БД удаленным персональным компьютерам.

8. **Контроль за доступом к данным**. Доступ к данным могут осуществлять только зарегистрированные в СУБД пользователи в соответствии с назначенными администратором СУБД им правами.

2.2 Компоненты СУБД

СУБД – это сложный вид программного обеспечения, над созданием которого работают большие коллективы высококвалифицированных программистов. На современном рынке программного обеспечения конкурирует около трех десятков коммерческих СУБД. Из малых систем, рассчитанных на одного пользователя, сегодня наибольшей популярностью пользуются Microsoft Access и SQLite. В перечень получивших широкое признание многопользовательских реляционных СУБД входят: Oracle, SQL Server компании Microsoft, InterBase, Firebird, MySQL, PostgreSQL, Informix и т.д.

Для того чтобы СУБД оказалась в состоянии предоставлять минимальный набор из рассмотренных выше восьми сервисов, она должна состоять из набора компонент, представленных на рисунке 6.

Над верхним уровнем системы расположены потребители услуг СУБД:

– администратор БД;

– программисты;

– конечные пользователи.

Администратор БД отвечает за планирование и физическую реализацию проекта. Он создает основные объекты БД, определяет правила поддержания целостности и непротиворечивости данных, управляет политикой безопасности, анализирует процесс эксплуатации проекта, контролирует производительность системы – поддерживает жизнедеятельность БД.

В распоряжении администратора имеются разнообразные средства проектирования БД, эти средства могут входить в состав СУБД в виде дополнительных программных модулей, а могут быть поставлены и сторонними разработчиками.

Программисты совместно с администратором трудятся над физическим созданием проекта БД. Но прикладных программистов в большей степени интересует не сама концепция проекта БД, а способы донесения этой концепции до конечного пользователя.

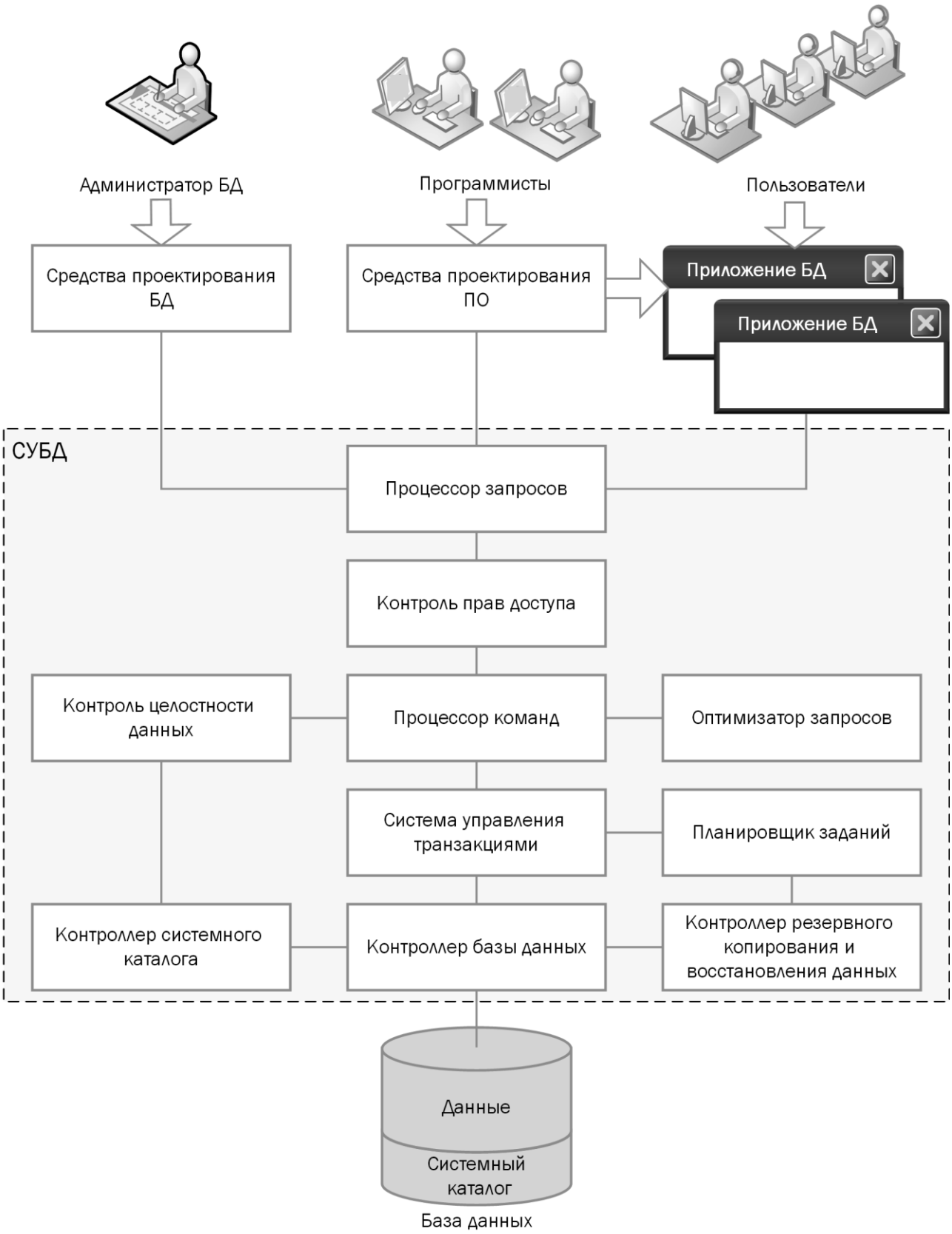


Рис. 6 – Обобщенная структура СУБД

Поэтому область интересов программистов смещена в сторону разработки клиентских приложений и отчетов. Основным инструментом прикладного программиста выступают многочисленные среды проектирования, как правило, не ниже 4-го поколения.

Основным потребителем услуг СУБД выступает обычный *пользователь* – в его интересах создаются БД и прикладное программное обеспечение.

Основным средством общения между людьми и базой данных выступает структурированный язык запросов SQL. Поэтому средства проектирования БД, ПО и клиентские приложения БД отправляют в адрес СУБД *инструкции на языке SQL*. Эти команды поступают на процессор запросов, который преобразует их в набор низкоуровневых команд, понятных ядру СУБД.

К базе данных могут обращаться различные категории пользователей, от руководителя предприятия до случайного посетителя. Именно поэтому в составе большинства СУБД имеется модуль, отвечающий за *контроль прав доступа*. В простейшем случае модуль следит за тем, чтобы к БД присоединялись только авторизованные пользователи, для этого полученные во время регистрации пароль и логин потенциального клиента сверяются с хранящимися в системном каталоге учетными записями. В современных многопользовательских СУБД система безопасности значительно сложнее и включает в себя комплекс программных и аппаратных средств защиты.

Убедившись, что инструкция поступила от доверенного лица, модуль контроля прав доступа передает ее в распоряжение *процессора команд*. В первую очередь процессор убеждается, что поступившая команда не противоречит ограничениям целостности данных. Это зона ответственности модуля контроля целостности данных. На время проверки команды на соответствие ограничениям целостности доменов, сущностей и связей задействуется *контроллер системного каталога*.

Именно он имеет возможность собирать метаданные, в которых прячется техническое описание БД. Поняв, что угрозы целостности нет, процессор передает команду оптимизатору запросов. Задача оптимизатора – найти

наиболее эффективный способ выполнения поступивших команд. Наконец, оптимизированная команда компилируется и передается *системе управления транзакциями*. Система управления транзакциями, во-первых, отвечает за полное и корректное выполнение блока команд и, во-вторых, совместно с планировщиком заданий обеспечивает параллельную многопользовательскую обработку данных.

Наконец, блок команд передается в распоряжение *контроллеру баз данных*. Задача модуля заключается в организации взаимодействия СУБД с файлами БД и файлами системного каталога. При этом для осуществления стандартных операций ввода-вывода задействуются возможности операционной системы.

Системный каталог

В отличие от систем, основанных на файлах, хранивших описание своих данных внутри исполняемых файлов, все современные СУБД обладают системным каталогом, в котором хранятся следующие сведения:

1) описание поддерживаемых типов данных;

2) описание развернутых БД (схемы данных) и входящих в них объектов (домены, таблицы, представления и т. д.);

3) сведения об ограничениях целостности;

4) имена и права пользователей, имеющих доступ к данным;

5) разнообразная статистика.

Благодаря системному каталогу легко определить смысл данных, что (в отличие от систем, основанных на файлах) позволит пользователям понять назначение и состав БД.

Обычно системный каталог физически реализуется в виде отдельной базы данных с системными таблицами по умолчанию, скрытыми от обычных пользователей.

2.3 Архитектурные решения доступа к базе данных

Во времена доминирования больших вычислительных машин доступ к самым первым БД базировался на принципе телеобработки. База данных и СУБД размещалась в памяти центральной ЭВМ. К этой ЭВМ подключались терминалы, которые не обладали собственными вычислительными возможностями), и вся обработка данных осуществлялась только в процессоре центральной ЭВМ. Системы на основе телеобработки доминировали до начала 1970-х годов, но с изобретением микропроцессоров стали постепенно уступать свои позиции и к сегодняшнему дню практически не используются.

Файл-сервер

Развитие микропроцессорной техники в 70-х годах прошлого века привело к появлению сравнительно дешевых микро-ЭВМ.

Теперь руководители предприятий вместо покупки одной большой дорогостоящей ЭВМ начали приобретать несколько менее производительных, но вполне приемлемых для решения типовых задач, стоящих перед их коллективами. Микро-ЭВМ объединялись в простейшие одноранговые локальные сети, в которых каждая из машин обладала равными правами со своими соседями.

Одна из ЭВМ (с накопителями на жестких магнитных дисках наибольшего размера) назначалась ***файловым сервером.*** На выданной в совместное пользование сетевой папке сервера, кроме обычных документов, размещали и файлы баз данных (рис. 7).

Для того чтобы этой БД могла воспользоваться какая-нибудь из рабочих станций, она обращалась к файловому серверу, перекачивала все файлы БД в свою память, вносила правки и возвращала файлы на прежнее место.

Такой способ многопользовательского доступа к БД обладал всего одним ***преимуществом*** – простотой реализации.

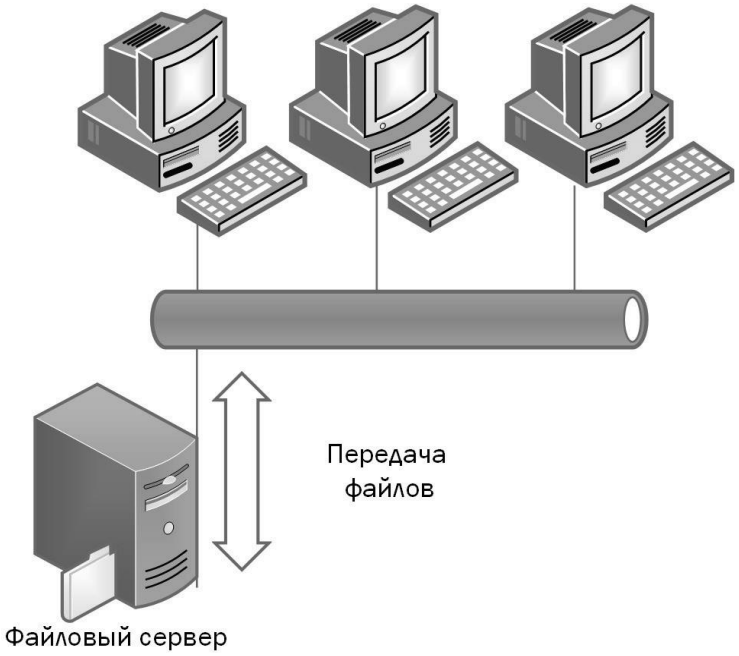


Рис. 7 – Архитектура «файл-сервер»

Недостатки системы:

– на каждой рабочей станции необходимо устанавливать полную копию СУБД;

– во время работы с данными сетевой трафик возрастает до пиковых значений;

– управление параллельностью обработки данных и поддержку целостности реализовать было практически невозможно, в то время как один пользователь сохранял свою порцию данных, другой уже вносил в нее изменения.

Клиент-сервер

Факт появления клиент-серверных систем тесным образом связан с парадигмой открытых систем, активно эволюционирующих с начала 1980-х годов. Программисты пришли к выводу о необходимости распределения задач между элементами системы, в данном случае между двумя типами процессов, которые могут выполняться на различных объединенных в вычислительную

сеть компьютерах. Сервер отвечает за предоставление каких-либо услуг клиентскому процессу. Клиент запрашивает у сервера определенные услуги и ресурсы.

В клиент-серверных системах БД размещается на отдельном наиболее производительном компьютере, на этом же компьютере разворачивается сервер (это и есть СУБД). На клиентских станциях достаточно установить сравнительно нетребовательное к ресурсам пользовательское ПО и настроить сетевой доступ к серверу СУБД. Работа клиент-серверных систем принципиально отличается от работы в системах «файл-сервер».

Теперь вместо перекачки файлов с БД клиентский компьютер отправляет серверу запрос, построенный на основе языка SQL. Получив и обработав инструкцию SQL, сервер возвращает клиентскому компьютеру результаты ее выполнения, например выборку определенных данных (рис. 8).

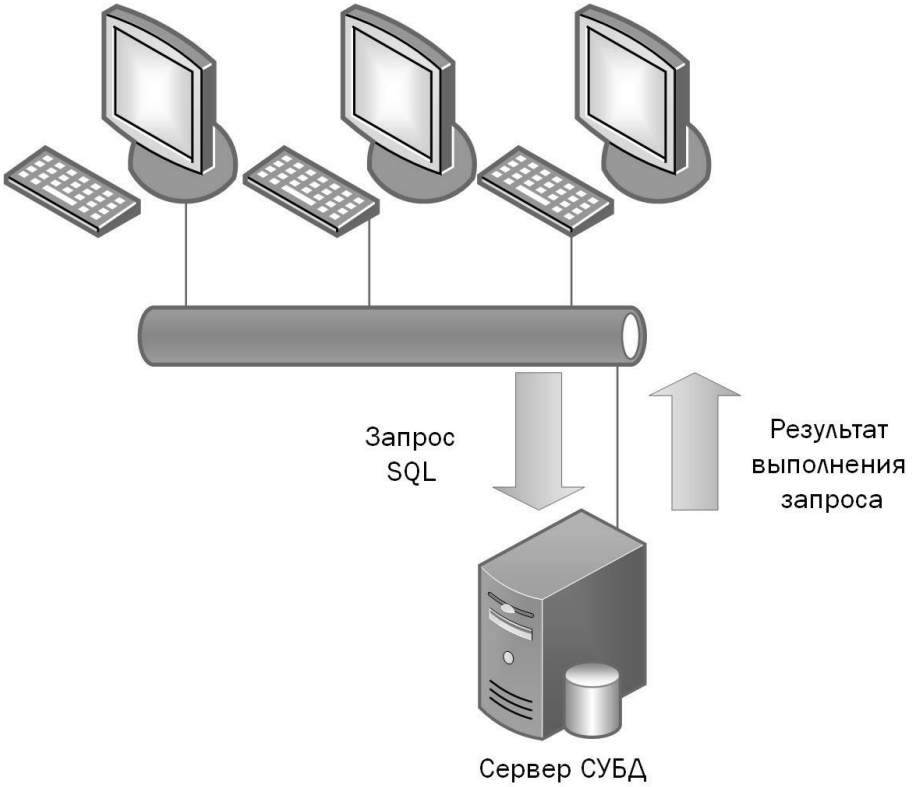


Рис. 8 – Архитектура «клиент-сервер»

Уже несколько десятилетий клиент-серверное построение БД является доминирующим. Причин тому несколько:

1) значительно повышается доступность БД. Сервер представляет собой открытую систему, поэтому клиентские компьютеры могут функционировать под управлением различных операционных систем и с различным ПО;

2) выделенный сервер СУБД в состоянии обеспечить параллельную многопользовательскую обработку данных;

3) основные правила поддержки целостности и непротиворечивости данных описываются на одном сервере СУБД, клиентские приложения никаким образом не в состоянии обойти эти правила;

4) экономно расходуется пропускная способность компьютерных сетей;

5) благодаря централизованному хранению данных сравнительно просто поддерживать единые для всех правила безопасности БД;

6) наличие стандарта на основной язык общения SQL обеспечивает широкие возможности доступа к серверу БД из программного обеспечения различных производителей;

7) упрощены вопросы обслуживания и администрирования БД.

Предусмотрено несколько подходов к распределению обязанностей между сервером и клиентом. В простейшем случае (рис. 9) работает ***модель тонкий клиент (thin client)***. Это тот вариант, когда клиентское ПО максимально упрощено и представляет собой только интерфейсную часть, состоящую из форм ввода и просмотра данных. Тонкий клиент фактически отвечает лишь за презентацию данных, поэтому можно сказать, что он не умеет думать и вся программная логика сосредоточена на стороне сервера.

Есть и обратное решение – ***толстый клиент (fat client).*** В противовес своему тонкому собрату, толстый клиент старается максимально облегчить работу сервера (рис. 9). Например, реализует дополнительные бизнес-правила, производит сортировку полученных данных на клиентской стороне, выполняет вспомогательные расчеты, проверяет синтаксис инструкций SQL и т. п.

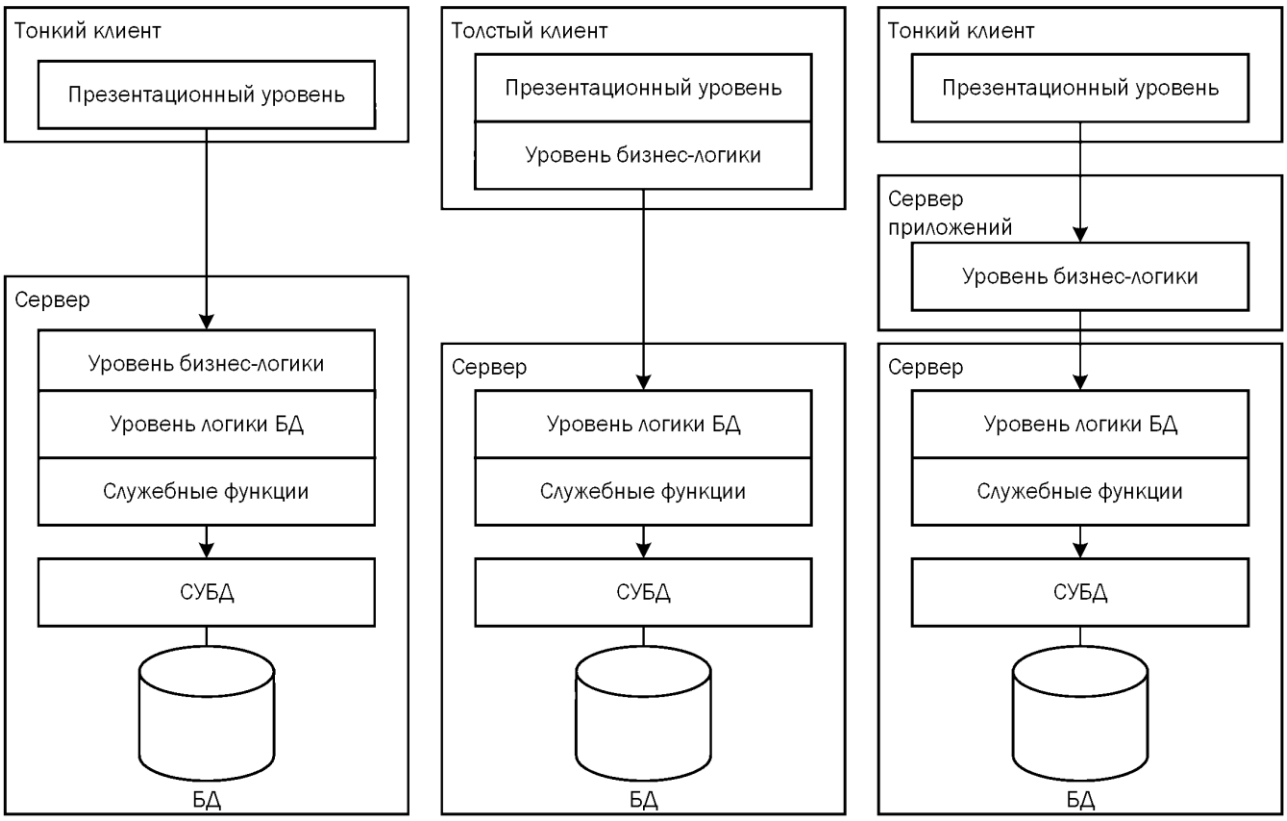


Рис. 9 – Модели распределения функций между сервером и клиентом

Проектирование полнофункционального толстого клиентского приложения значительно сложнее, чем разработка его упрощенного собрата. Но в конечном счете увеличивается производительность, а это весьма важно в больших многопользовательских БД.

К настоящему времени существует несколько сотен СУБД, предназначенных для работы по модели клиент-сервер. В России наибольшей популярностью пользуются MySQL, Oracle, SQL Server компании Mirosoft, InterBase компании Embarcadero, PostgreSQL, Informix компании IBM, NetWare SQL фирмы Novell и т. д.

Многоуровневые решения

Предусмотрены и более сложные модели распределения функций между сервером и клиентом. Например, между сервером СУБД и клиентом может появиться еще одно дополнительное звено – ***сервер приложений*** (рис. 9).

Сервер приложений обычно отвечает за соблюдение бизнес-правил БД, для этого реализуется несколько прикладных функций, которые представляются клиенту в виде отдельных служб. Клиент не может обратиться к СУБД напрямую, вместо этого он запрашивает интересующий его сервис у сервера приложений.

При создании промежуточного звена между клиентом и сервером разработчики часто пользуются технологией «Архитектура брокера общих объектных запросов» (Common Object Request Broker Architecture, CORBA). Технология CORBA создавалась специально для систем с распределенной обработкой информации.

Несмотря на то что трехзвенная архитектура позволяет создавать гибкие и универсальные решения, она используется значительно реже, чем двухзвенная. Причин тому несколько, но основная из них в том, что создание многоуровневых клиент-серверных проектов под силу только хорошо подготовленным разработчикам, как следствие подобные проекты стоят значительно дороже, по сравнению с классической архитектурой «клиент-сервер».

Распределенная система

Системы управления распределенными базами данных (СУРБД) – одно из направлений развития современных технологий хранения данных. Подобные системы предполагают организацию работы с данными, распределенными между несколькими серверами, которые, в свою очередь, могут быть удалены друг от друга на значительные расстояния (рис. 10).

Любой из серверов должен обладать возможностью обрабатывать как локальные запросы пользователей в своей подсети, так и слаженно работать с внешними запросами, поступающими из других подсетей. В свою очередь, клиент вправе получать одновременный доступ к интересующим его данным, физически размещенным на разных серверах (клиент вообще может полагать, что он работает с единой, а не распределенной БД).

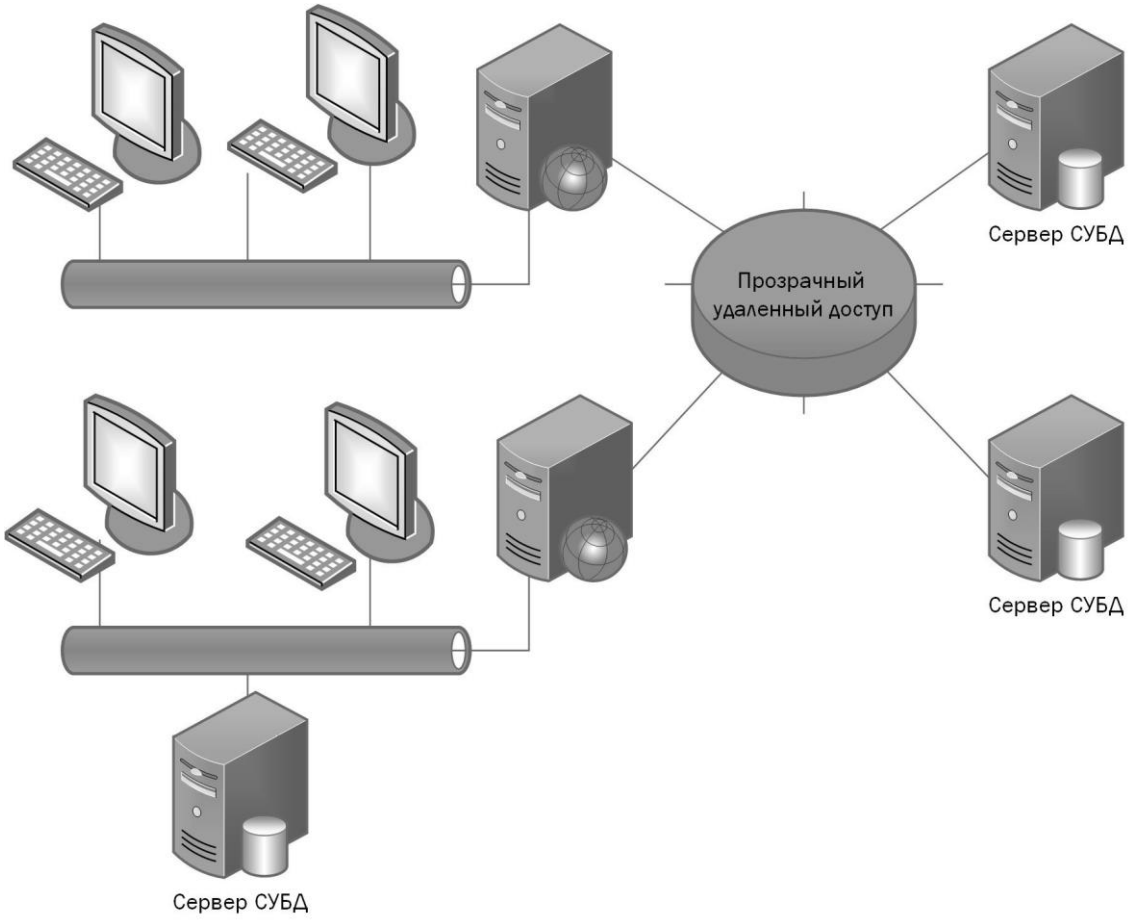


Рис. 10 – Архитектура распределенной обработки

Распределенные системы могут включать как однотипные (гомогенные), так и разнотипные (гетерогенные) БД. СУБД обслуживают фрагменты БД, между которыми может осуществляться частичная или полная репликация данных. Кроме того, могут использоваться специальные серверы, предназначенные для управления всей системой (например, на них может содержаться глобальный системный каталог, схема фрагментации данных между СУБД, средства обработки распределенных запросов и т. д.).

Контрольные вопросы:

1. Дайте определение СУБД.
2. Какие различия имеются между системой файлов и СУБД?
3. Каким образом классифицируются СУБД?

4. Какие основные функции должна выполнять СУБД?

5. Что понимается под термином метаданные?

6. Опишите назначение компонентов СУБД.

7. Какие архитектурные решения доступа к БД вам известны?

8. Почему архитектура файл-сервер не подходит для многопользовательских БД?

9. Как могут распределяться задачи между клиентом и сервером БД?

10. Проанализируйте принципы организации СУБД на основе архитектура распределенной обработки данных.

ТЕМА 3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Любая БД представляет собой хранилище структурированных данных, которое для полноценной реализации производственно-хозяйственных функций должно быть интегрировано в состав более сложной надсистемы, называемой информационной.

Информационная система (Information System, IS) – это система, предназначенная для сбора, корректировки и распространения информации внутри организации и объединяющая в своем составе персонал, оборудование, базы данных и программное обеспечение. Все элементы ИС объединены общей целью – обеспечить поддержку принятия решений по эффективному управлению предприятием, в котором развернута эта система (рис. 11).

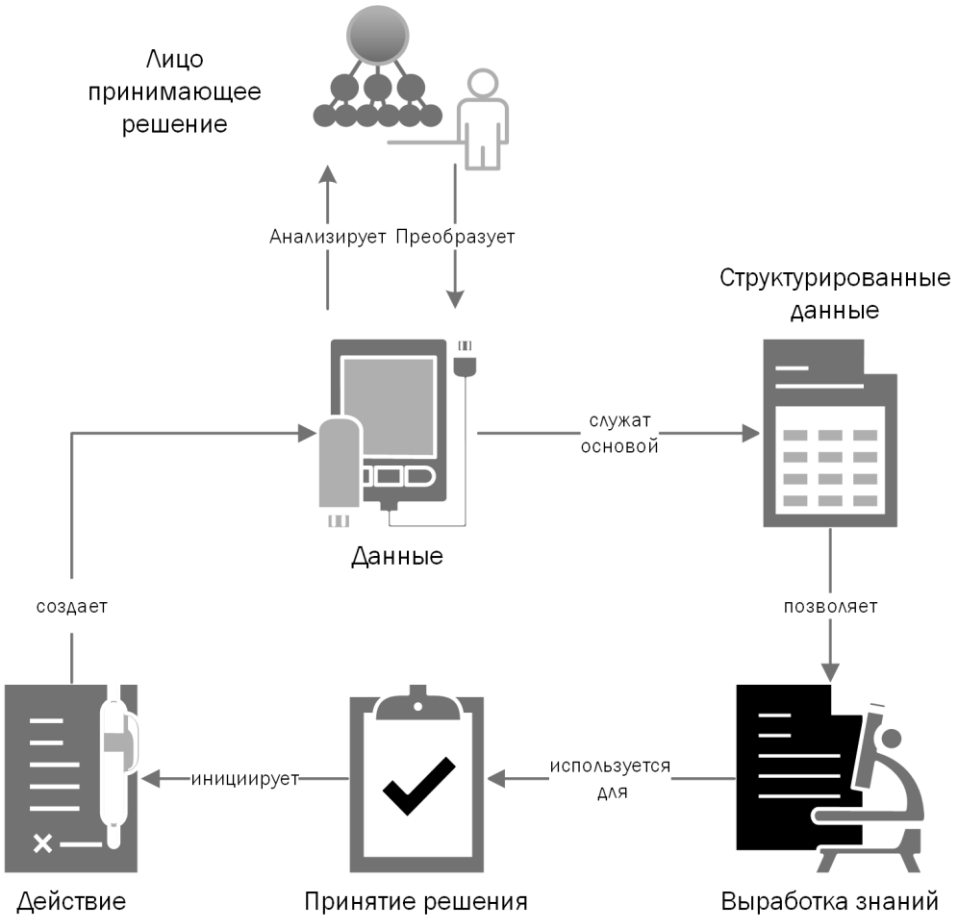


Рис. 11 – Цикл преобразования информации

Цикл преобразования информации действует непрерывно, обеспечивая функционирование предприятия. Лицо, принимающее решение (руководитель, директор, совет директоров), анализирует и преобразует имеющиеся в его распоряжении данные о состоянии дел в его предприятии. В свою очередь, данные выступают основой для построения специально обработанных (структурированных) данных, в данном случае – базы данных. Благодаря БД данные представляются и интерпретируются в удобном для анализа виде. Структурированные данные позволяют вырабатывать новые знания, которые будут использованы для принятия очередного управляющего решения. Появление решения приводит к выполнению определенного действия, которое порождает новые данные.

Создание информационной системы в общем случае включает этапы:

1) планирования системы;

2) анализа задач системы и требований к системе;

3) проектирования системы;

4) реализации и ввода в эксплуатацию;

5) сопровождения.

БД является фундаментом информационной системы, однако следует помнить, что в состав информационной системы также входит программное обеспечение поддержки БД, аппаратное обеспечение, прикладное ПО и эксплуатирующий систему персонал.

При возникновении вопроса необходимости информатизации и автоматизации процессов на предприятии, возникает вопрос покупки готового программного продукта, либо создания собственного. Есть существенная разница между универсальным программным продуктом стороннего производителя и ПО, разработанным специально для компании. Первый продукт необходимо адаптировать к условиям конкретной компании, второй сделан под запросы и нужды предприятия. Кроме того, наличие в распоряжении компании своего собственного разработчика позволит ей оперативно парировать все сбои в работе БД и быстро вносить изменения и доработки.

3.1 Жизненный цикл базы данных

Жизненным циклом БД называют период от момента появления идеи создания БД до момента завершения его поддержки разработчиком или организацией, выполнявшей сопровождение.

Состав процессов жизненного цикла БД (рис. 12) регламентируется международным стандартом ISO/IEC 12207:2008 Systems and software engineering – Software life cycle processes (Системная и программная инженерия – Процессы жизненного цикла).

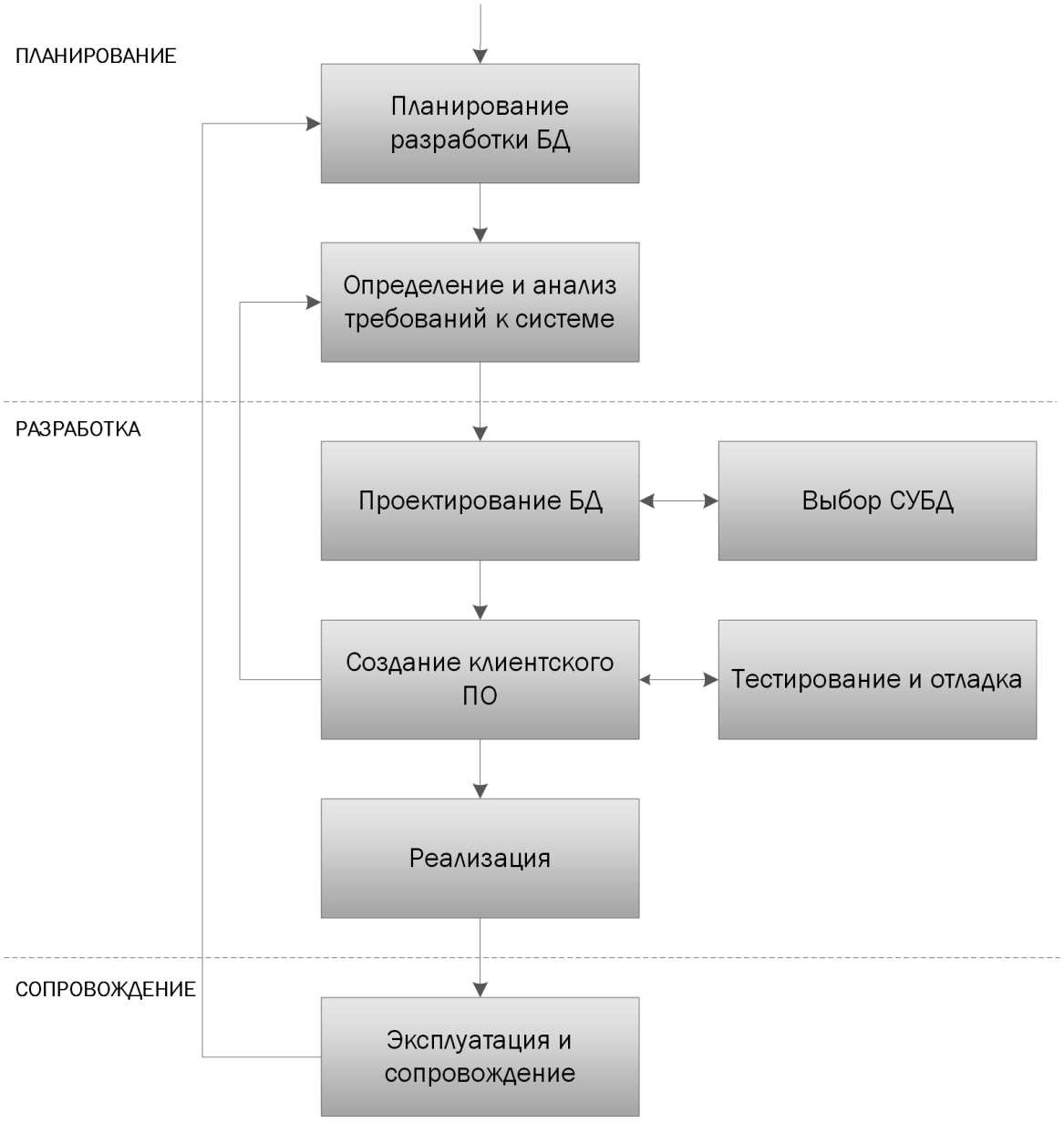


Рис. 12 – Жизненный цикл проекта БД

3.2 Планирование разработки базы данных

Содержание данного этапа – разработка стратегического плана, в процессе которой осуществляется предварительное планирование конкретной системы управления базами данных, которое заключается в определении трех основных компонентов: объема работ, ресурсов и стоимости проекта.

Важной частью разработки стратегического плана является проверка осуществимости проекта, состоящая из нескольких частей:

– проверка технологической осуществимости. Она состоит в выяснении вопроса, существует ли оборудование и программное обеспечение, удовлетворяющее информационным потребностям фирмы;

– проверка операционной осуществимости – выяснение наличия экспертов и персонала, необходимых для работы БД;

– проверка экономической целесообразности осуществления проекта. При исследовании этой проблемы весьма важно дать оценку ряду факторов, в том числе и таким:

– целесообразность совместного использования данных разными отделами;

– величина риска, связанного с реализацией системы базы данных;

– ожидаемая выгода от внедрения подлежащих созданию приложений;

– время окупаемости внедренной БД;

– влияние системы управления БД на реализацию долговременных планов организации.

3.3 Этап определения и анализа требований к системе

На этом этапе должна быть собрана вся информация, необходимая для проектирования БД.

Вне зависимости от масштабов БД на данном этапе необходимо получить следующую информацию:

1) цели и задачи компании;

2) организационно-штатная структура компании;

3) модель бизнес-процесса компании.

4) выделение границ и возможностей проекта;

5) анализ смежных бизнес-процессов с целью выявления перспектив дальнейшего совершенствования БД.

Цели и задачи компании необходимы для формирования понимания направления основной деятельности компании. Зная целевую функцию, можно, пусть даже пока на упрощенном уровне, предсказать, что именно захочет получить от будущей БД заказчик.

Знания организационной структуры предприятия необходимы при определении информационных потоков внутри компании. Это объясняется разнообразием запросов конечных пользователей проектируемой БД: кому-то нужны отчеты, кому-то аналитика, кто-то хочет владеть всеми данными, кому-то достаточно знать лишь детали.

При составлении модели бизнес-процессов компании необходимо вовлечь в процесс проектирования БД максимальное число специалистов компании. Информация может быть получена следующими путями:

– опрос основных специалистов компании;

– анализа обязанностей сотрудников;

– изучение документов на рабочих местах, в отделах и службах компании, в особенности документов, претендующих на роль отчетных (ведомости, накладные, заказы, заявки и т. д.);

– наблюдение за процессом функционирования компании, особенно в области документооборота;

– анкетирование сотрудников компании.

После построения модели бизнес-процессов компании необходимо найти компромисс между излишней детализацией, с одной стороны, и недопустимым упрощением – с другой. Очерчивая границы проекта, необходимо учесть возможности и перспективы его дальнейшего развития.

Логическим завершением этапа определения и анализа требований к системе должен стать проект технического задания на разработку БД. Указанный проект согласовывается с участвующими в проекте руководителями основных отделов и служб компании и утверждается руководством компании (заказчиком БД). В окончательное техническое задание проект превратится в середине этапа проектирования БД [11].

3.4 Этап проектирования БД

Специалисты разделяют этот этап на три фазы (рис. 13):

1) концептуальное проектирование;

2) логическое проектирование;

3) физическое проектирование.

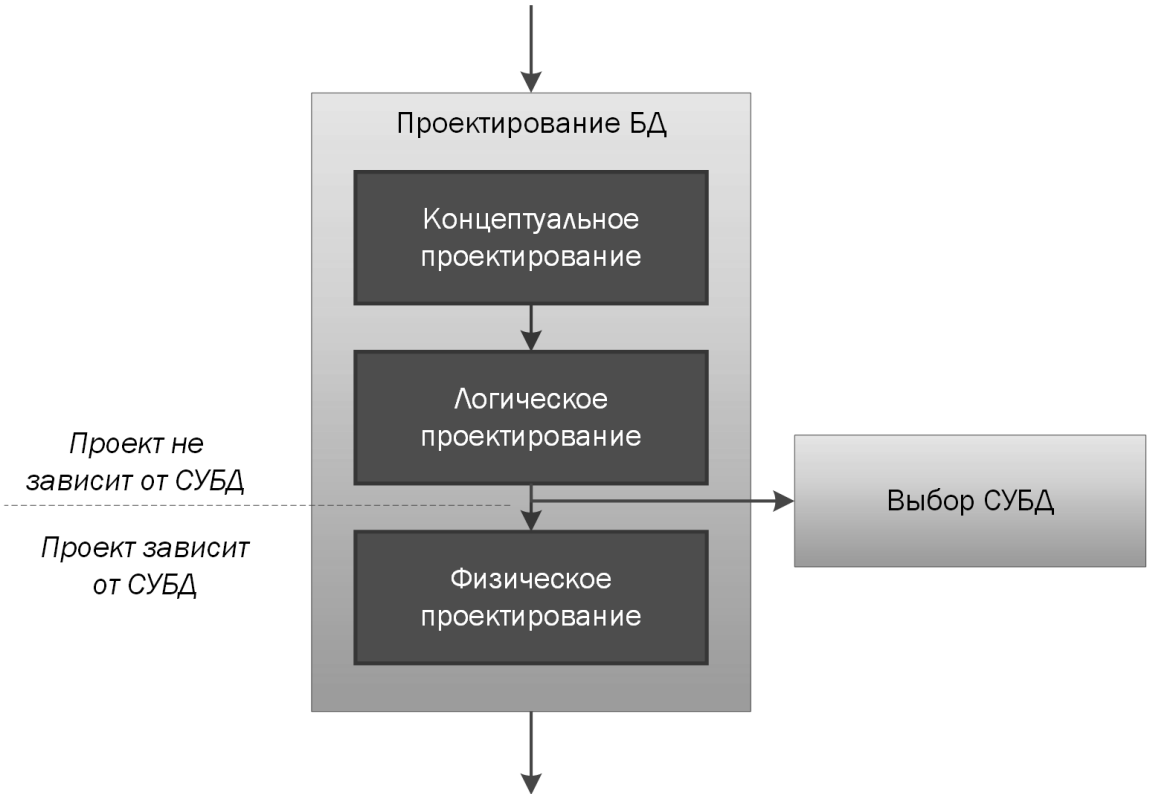


Рис. 13 – Фазы проектирования базы данных

Во время ***концептуального проектирования*** окончательно формируется замысел будущей базы данных, но без учета любых физических аспектов ее реализации. На этой ступени проектирования разработчика пока не интересует ни конкретная СУБД, на которой позднее развернется БД, ни используемый для создания приложений язык программирования, ни особенности аппаратной платформы. Основной интерес направлен на создание общей модели, отражающей представления будущих пользователей БД об автоматизируемом участке компании (складе, бухгалтерии, отделе кадров, производственных цехах и т. п.). Вся необходимая для этого информация уже должна быть собрана на предыдущем этапе жизненного цикла БД.

Основным средством построения концептуальной модели БД выступает модель «сущность-связь» или родственные ей модели. Задачи их построения заключается в наглядном представлении данных, подлежащих хранению в БД. В построении общей концептуальной модели данных выделяют ряд этапов:

– выделение локальных представлений, соответствующих обычно относительно независимым данным. Каждое такое представление проектируется как подзадача;

– формулирование сущностей, описывающих локальную предметную область проектируемой БД, и описание атрибутов, составляющих структуру каждой сущности;

– выделение ключевых атрибутов;

– спецификация связей между сущностями. Удаление избыточных связей;

– анализ и добавление неключевых атрибутов;

– объединение локальных представлений.

На завершающей стадии концептуального проектирования разработчик БД должен проверить адекватность полученной модели путем обсуждения полученных результатов с сотрудниками компании. При обнаружении несоответствий в модель вносятся исправления. Процесс сверки прекращается только после того, когда все пользователи подтвердят корректность концептуальной модели.

Результатом фазы концептуального проектирования станет ER-модель будущей БД, включающая в себя описание:

– типов сущностей;

– связей между типами сущностей;

– атрибутов (желательно с предварительным описанием доменов и ограничений);

– первичных ключей.

На этапе ***логического проектирования*** разработчик уточняет все требования, выявленные на концептуальной стадии проектирования, и стремится несколько упростить решение (при этом не снижая его функциональные возможности). Для этого предварительная ER-модель проверяется с помощью правил *нормализации*. В результате получаем неизбыточные реляционные таблицы, свободные от присущих ненормализованным данным аномалий вставки, редактирования и удаления. Помимо нормализации, на логическом этапе осуществляются следующие действия:

– уточняются ограничения на данные;

– определяются домены данных;

– вводятся бизнес-правила и корпоративные ограничения целостности;

– определяется местоположение будущих таблиц (в случае если речь идет о распределенной БД).

Представленная логическая модель сверяется с будущими пользователями БД и заказчиком проекта, вносятся уточнения и исправления.

На этапах концептуального и логического проектирования разработчики обычно пользуются одной из двух стратегий проектирования БД: *стратегией восходящего проектирования* (bottom-up design) или *стратегией нисходящего проектирования* (top-down design).

Восходящий подход обычно применяется для сравнительно небольших проектов. Суть метода заключается в том, что проектировщик совместно с заказчиком БД строит полный список атрибутов (столбцов таблиц),

подлежащих хранению. Позднее атрибуты группируются в типы сущностей, которые попадают в модель.

Нисходящая стратегия лучше подходит для средних и больших проектов. Здесь проектирование начинается с выявления основных типов сущностей, и только затем сущности приобретают атрибуты.

По завершении фазы логического проектирования совместно с заказчиком создается окончательная версия технического задания на БД. В техническом задании закрепляются требования к проекту, утверждается логическая модель БД, описываются необходимые отчеты и т. п. После разработки технического задания между разработчиком БД и заказчиком заключается юридический договор на разработку БД, в котором, помимо всего прочего, окончательно утверждаются стоимость и сроки выполнения проекта.

Переход к ***физической фазе проектирования*** БД производится после выбора целевой СУБД, именно она определяет особенности будущего программного продукта. Поэтому все последующие фазы проектирования БД и этапы жизненного цикла БД приобретают зависимость от СУБД.

Физическое проектирование – это уточнение решения с учетом имеющихся в наличии разработчика технологий, возможности реализации и требуемой производительности. Во время физического проектирования задачей проектировщика становится перенос логической модели на платформу целевой СУБД, что предусматривает:

– создание таблиц и связей между ними;

– назначение вторичных индексов таблиц;

– разработку представления;

– реализацию бизнес-логики БД (в первую очередь с помощью триггеров и хранимых процедур);

– определение функциональных характеристик транзакций;

– внедрение механизмов защиты (авторизация пользователей, назначение правил доступа к данным и т.д.).

Полученная БД документируется, в особенности в части:

– определения пользовательских типов данных;

– описания таблиц и связей, порядка поддержки бизнес-логики;

– назначения и порядка вызова хранимых процедур и триггеров.

3.5 Этап создания клиентского программного обеспечения

Завершив работу над формированием физической структуры БД, проектировщик переходит к этапу разработки приложений, предназначенных для работы с БД.

Задачей разрабатываемого прикладного ПО является, с одной стороны – предоставление интуитивно понятного интерфейса конечному пользователю, а с другой – обеспечение прозрачного для пользователя взаимодействия с БД. В самом общем случае приложение баз данных должно выполнять следующие функции:

– получать доступ к БД;

– читать, добавлять, редактировать и удалять данные;

– представлять полученные данные в требуемом пользователем виде (формы, отчеты, многомерное представление и т. д.);

– поддерживать целостность данных, определять дополнительную бизнес-логику и ограничения на данные;

– обеспечивать требуемую безопасность данных.

В качестве инструментальных средств, применяемых для написания прикладного ПО, сегодня используются языки программирования 4-го поколения.

Проектирование транзакций.

Транзакция – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Транзакция может состоять из нескольких операций, однако с точки зрения пользователя эти операции представляют собой единое целое, переводящее базу данных из одного непротиворечивого состояния в другое. Реализация транзакций опирается на

тот факт, что СУБД способна обеспечивать сохранность внесенных во время транзакции изменений в БД и непротиворечивость базы данных даже в случае возникновения сбоя. Проектирование транзакций заключается в определении:

– данных, которые используются транзакцией;

– функциональных характеристик транзакции;

– выходных данных, формируемых транзакцией;

– степени важности и интенсивности использования транзакции.

Проектирование пользовательского интерфейса.

Интерфейс должен быть удобным и обеспечивать все функциональные возможности, предусмотренные в спецификациях требований пользователей.

Специалисты рекомендуют при проектировании пользовательского интерфейса использовать следующие основные элементы и их характеристики:

– содержательное название;

– ясные и понятные инструкции;

– логически обоснованные группировки и последовательности полей;

– визуально привлекательный вид окна формы или поля отчета;

– легко узнаваемые названия полей;

– согласованную терминологию и сокращения;

– согласованное использование цветов;

– визуальное выделение пространства и границ полей ввода данных;

– удобные средства перемещения курсора;

– средства исправления отдельных ошибочных символов и целых полей;

– средства вывода сообщений об ошибках при вводе недопустимых значений;

– особое выделение необязательных для ввода полей;

– средства вывода пояснительных сообщений с описанием полей;

– средства вывода сообщения об окончании заполнения формы.

3.6 Этап тестирования и отладки

Без тестирования невозможно гарантировать заказчику правильную работу созданного продукта. Поэтому, вне зависимости от степени сложности БД, для каждого проекта необходимо разработать исчерпывающий план тестирования, распространяющийся на все основные функции БД и прикладного ПО [12-15].

Тестирование – это процесс выполнения программы с целью обнаружения ошибок. Оно обеспечивает:

– обнаружение ошибок;

– демонстрацию соответствия функций БД и клиентского ПО их назначению;

– демонстрацию реализации требований к характеристикам БД и клиентского ПО.

Косвенно тестирование осуществляет проверку ряда показателей надежности.

Различают два принципа тестирования ПО:

1) структурное тестирование;

2) функциональное тестирование.

Структурное тестирование (тестирование «белого ящика», т. к. полностью доступен исходный код программы) основано на анализе управляющей структуры программы. Лицо, ответственное за проведение тестирования, строит граф управления программой и формирует тестовые варианты для всех возможных маршрутов.

Во время структурного тестирования проводится:

1) модульное тестирование – тестированию подлежат минимальные компоненты ПО, например, запрос, процедура, триггер;

2) интеграционное тестирование, при котором модули ПО объединяются и осуществляется проверка на корректность взаимодействия между интегрируемыми компонентами.

Функциональному тестированию подвергается уже полностью откомпилированное программное изделие. На этом этапе исходный код вторичен, поэтому процесс называют тестированием «черного ящика». По своей сути функциональное тестирование ПО представляет собой эксплуатацию приложения в контролируемых условиях с последующим анализом полученных результатов. При этом проверяется работа приложения не только с нормальными, но и с ошибочными данными. Также во время тестирования следует изучить поведение БД и ПО в нештатных ситуациях.

Функциональное тестирование обычно включает в себя два этапа: альфа-тестирование и бета-тестирование. Во время *альфа-тестирования* имитируется реальная работа БД и прикладного ПО, но в качестве пользователей выступают штатные разработчики проекта. На этапе *бета-тестирования* программное обеспечение передается заказчику (или третьему лицу) с целью апробации работы и выявления не замеченных ранее ошибок в его работе.

Для оценки законченности и корректности выполнения приложения базы данных может использоваться несколько различных стратегий тестирования:

– нисходящее тестирование;

– восходящее тестирование;

– тестирование потоков;

– интенсивное тестирование.

Нисходящее тестирование начинается на уровне подсистем с модулями, которые представлены заглушками, т. е. простыми компонентами, имеющими такой же интерфейс, как модуль, но без функционального кода. Каждый модуль низкого уровня представляется заглушкой. Постепенно все программные компоненты заменяются фактическим кодом и после каждой замены снова тестируются.

Восходящее тестирование выполняется в противоположном направлении по отношению к нисходящему. Оно начинается с тестирования модулей на самых низких уровнях иерархии системы, продолжается на более высоких уровнях и заканчивается на самом высоком уровне.

Тестирование потоков осуществляется при тестировании работающих в реальном масштабе времени систем, которые обычно состоят из большого количества взаимодействующих процессов, управляемых с помощью прерываний. Стратегия тестирования потоков направлена на слежение за отдельными процессами.

Стратегия интенсивного тестирования часто включает серию тестов с постепенно возрастающей нагрузкой и продолжается до тех пор, пока система не выйдет из строя.

Все выявленные во время тестирования ошибки должны быть локализованы и исправлены, для этого осуществляется отладка ПО.

Отладка – это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения.

Различают три вида ошибок.

1. *Синтаксические ошибки* – ошибки, выявляемые при выполнении синтаксического и частично семантического анализа программы. Они автоматически фиксируются компилятором (транслятором, интерпретатором).

2. *Ошибки компоновки* – ошибки, обнаруженные компоновщиком при объединении модулей программы.

3. *Ошибки выполнения* – самая сложная для устранения категория ошибок. Ошибки выполнения обнаруживаются операционной системой, аппаратными средствами, лицами, осуществляющими тестирование, или просто конечным пользователем при работе с программой.

3.7 Этап реализации

На этапе реализации осуществляется выпуск окончательной версии программного продукта. Прежде чем проектировщик переходит к выпуску БД, необходимо убедиться в выполнении следующих условий:

– база данных и пользовательские приложения обладают всеми указанными в техническом задании функциональными возможностями;

– программное обеспечение успешно прошло тестирование, и все выявленные критические ошибки устранены;

– заказчик и проектировщик приняли совместное решение о том, что реализация всех элементов БД в целом завершена.

На завершающей стадии этапа реализации разработчик ПО готовит материалы, необходимые для обучения и технической поддержки пользователей и для сопровождения приложения.

3.8 Этап эксплуатации и сопровождения

Это заключительный этап жизненного цикла БД. В его начальной стадии осуществляется развертывание базы данных и прикладного ПО на сервере и клиентских станциях заказчика ПО. При необходимости администратор СУБД создает учетные записи пользователей базы данных. После этого проект полностью переходит под контроль заказчика.

Основная часть обязанностей по сопровождению БД возлагается на администратора БД, который осуществляет профилактическое обслуживание БД, восстанавливает систему после сбоев, управляет учетными записями пользователей, следит за производительностью системы, поддерживает безопасность БД, ведет системный аудит.

Поддержка БД предполагает разрешение проблем, возникающих в процессе эксплуатации БД и связанных как с ошибками реализации БД, так и с изменениями в самой предметной области, созданием дополнительных программных компонент или модернизацией самой БД.

Контрольные вопросы:

1. Проанализируйте роль и укажите место БД в цикле преобразования информации.

2. По каким признакам можно судить о необходимости создания (модернизации) БД на предприятии?

3. Какие ключевые этапы составляют жизненный цикл БД?

4. На какие вопросы следует получить ответы на этапе планирования разработки БД?

5. Какую информацию необходимо собрать, приступая к этапу определения и анализа требований к системе?

6. Каким образом следует осуществлять сбор требований пользователей к проектируемой БД?

7 Какие фазы проектирования БД вам известны?

8. Какие виды тестирования ПО необходимо провести перед вводом БД в эксплуатацию?

9. С какими категориями ошибок приходится сталкиваться во время отладки?

10. Почему эксплуатацию и сопровождение нельзя считать заключительным этапом жизненного цикла БД?

ТЕМА 4 МОДЕЛИ ДАННЫХ

Можно по-разному характеризовать понятие модели данных. С одной стороны, модель данных – это способ структурирования данных, которые рассматриваются как некоторая абстракция в отрыве от предметной области.

С другой стороны, модель данных – это инструмент представления концептуальной модели предметной области и динамики ее изменения в виде базы данных.

Модель является представлением реального мира объектов и событий, а также существующих между ними связей. Это некоторая абстракция, в которой акцент делается на самых важных и неотъемлемых аспектах деятельности организации, а все второстепенные свойства игнорируются. Модель должна отражать основные концепции, представленные в таком виде, который позволит проектировщикам и пользователям базы данных обмениваться конкретными и недвусмысленными мнениями о роли тех или иных данных в организации. Модель данных можно рассматривать как сочетание трех указанных ниже компонентов:

– *структурная часть* – набор правил, по которым может быть построена база данных;

– *управляющая часть*, определяющая типы допустимых операций с данными (сюда относятся операции обновления и извлечения данных, а также операции изменения структуры базы данных);

– *набор ограничений* поддержки целостности данных, гарантирующих корректность используемых данных.

Цель построения модели данных заключается в представлении данных в понятном виде. Если такое представление возможно, то модель данных можно легко применить при проектировании базы данных.

4.1 Эволюция моделей реализации данных

Становление и развитие теории баз данных можно разделить на несколько ключевых этапов, каждый из которых связан с появлением новых технологий структуризации, хранения и обработки информации. Эволюция моделей реализации данных представлена на рисунке 14.

Сегодня системы, основанные на файлах, практически не используются, исключение составляют состоящие из одного-двух файлов данных небольшие по числу записей хранилища.

Появление первых моделей БД (сетевая и иерархическая модели) заместило файловые системы, этот условный «дореляционный» период продолжался с середины 60-х по первую половину 70-х годов XX века до публикации появления теории Э. Кодда о реляционных банках данных.

Рис. 14 – Эволюция моделей реализации данных

4.2 Иерархическая модель

Суть иерархического хранения данных заключается в применении инвертированной древообразной структуры. В наивысшей точке располагался корень дерева, ниже – дочерние узлы (листья дерева). Все узлы связывались друг с другом благодаря сложной системе указателей. Каждый из узлов, в свою очередь, мог являться родительским по отношению к одному или нескольким нижерасположенным узлам, но у дочернего узла не может быть более одного родителя (рис. 15-16).

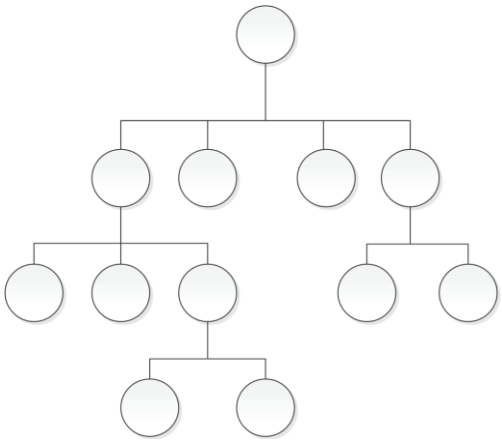


Рис. 15 – Иерархическая модель данных

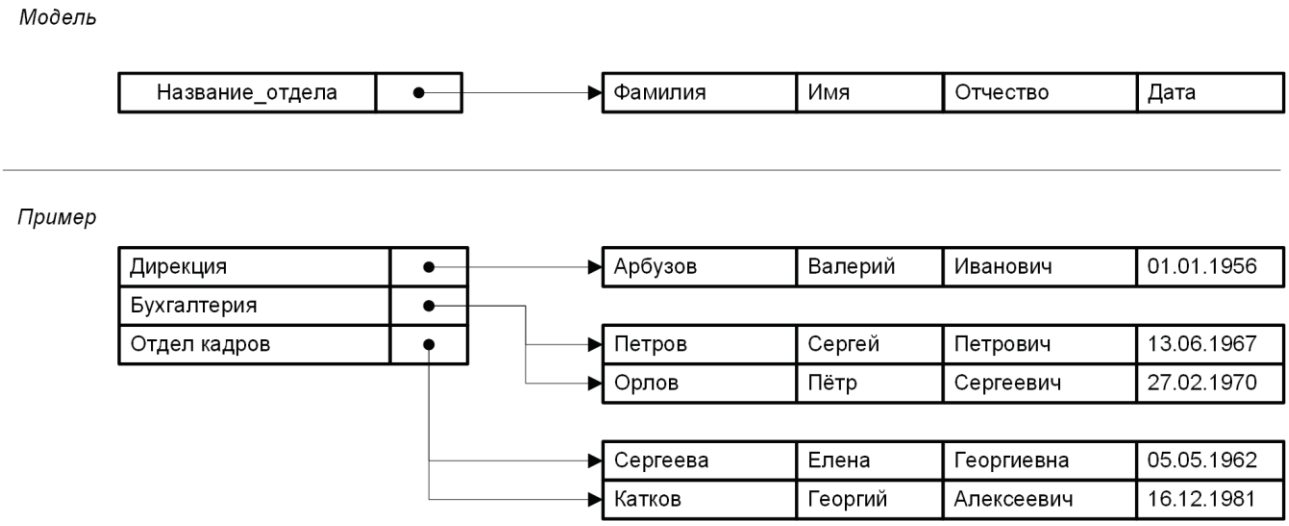


Рис. 16 – Пример работы иерархической БД

Таким образом, в иерархической модели реализована одна из наиболее часто встречающихся в реальном мире связей между сущностями – связь «один ко многим».

По сравнению с системами файлов иерархическая модель обладает следующими ***достоинствами:***

1) простота понимания структуры данных. Иерархическое построение данных интуитивно понятно, что существенно упрощает проектирование БД;

2) целостность данных. Иерархические БД представляли собой неразрозненные приложения и файлы. Все данные находились под контролем одной системы управления базами данных, которая не допускала некорректные действия с записями (например, удаление родительского узла, у которого оставались дочерние элементы);

3) независимость данных. Данные больше не принадлежат одному приложению. Наличие системного каталога позволяет работать с БД приложениям, умеющим читать метаданные;

4) безопасность данных. Контролируемый доступ к данным осуществляется с помощью СУБД, в которую закладывается предпочтительная политика безопасности;

5) иерархическая организация данных удобна при осуществлении поиска данных, запрашиваемые сведения уже сгруппированы по соответствующим ветвям дерева, и достаточно лишь спускаться от корня схемы к требуемому листу.

Однако, иерархическая модель имеет ряд ***недостатков***:

1) ограничения в организации отношений между сущностями. Иерархическая модель позволяет организовать последовательную связь «один ко многим» между данными, но не в состоянии реализовать отношения «многие ко многим»;

2) структурная зависимость. Иерархическая структура предполагала, что физически данные также станут храниться в виде дерева. Серьезное изменение

структуры (например, переподчинение узлов) могло привести к тому, что прикладные приложения теряли возможность навигации по данным;

3) сложность разработки прикладного программного обеспечения, разработчик которого должен знать особенности физического хранения данных, иначе он мог просто «заблудиться» в запутанной системе указателей;

4) иерархическая модель не была стандартизирована. Как следствие всегда существовала проблема переносимости данных между приложениями различных разработчиков.

4.3 Сетевая модель

В сетевой модели данных (СМД) элементарные данные и отношения между ними представляются в виде ориентированной сети (вершины – данные, дуги – отношения). БД, описываемая сетевой моделью, состоит из нескольких областей.

Область – это поименованная часть базы данных, в которой могут содержаться экземпляры записей и наборов или части наборов. Каждая область может обладать собственными уникальными физическими характеристиками. Области могут обрабатываться как по отдельности, так и вместе с другими областями.

Набор – это поименованная совокупность связанных записей. Набор может размещаться в одной или нескольких областях.

Сетевая БД состоит из набора записей и набора соответствующих связей. Сетевую модель можно представить как граф с записями в виде узлов графа и наборами в виде его ребер.

Для описания схемы сетевой БД используется две группы типов: ***«запись»*** и ***«связь».*** Тип «связь» определяется для двух типов «запись»: предка и потомка. Переменные типа «связь» являются экземплярами связей.

На форматирование связи особых ограничений не накладывается. Если в иерархических структурах запись-потомок могла иметь только одну запись-

предка, то в сетевой модели данных запись-потомок может иметь произвольное число записей-предков (свободных родителей).

Физическое размещение данных в базах сетевого типа может быть организовано практически теми же методами, что и в иерархических базах (рис. 17). Пример простой сетевой структуры показан на рисунке 18.

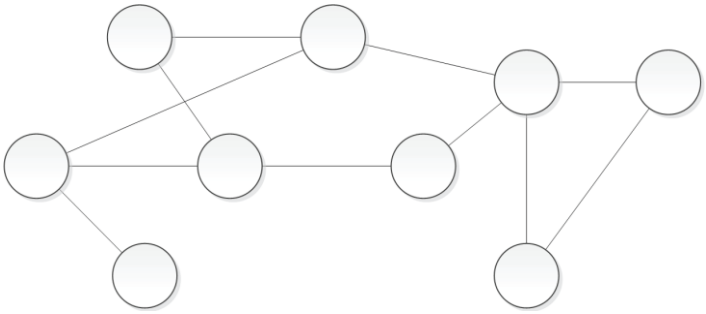


Рис. 17 – Сетевая модель данных

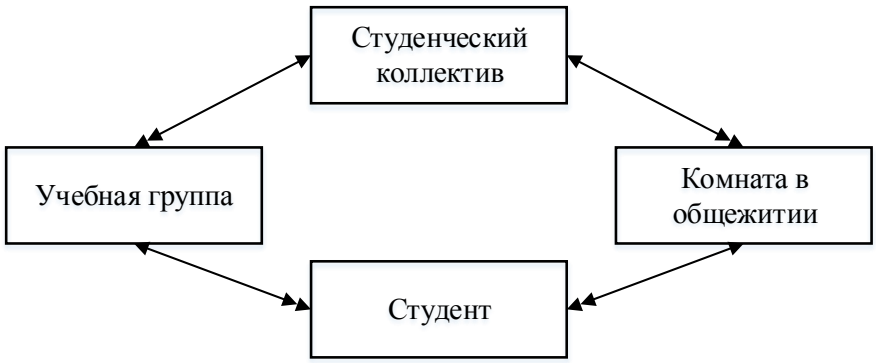


Рис. 18 – Пример сетевой модели данных

К числу *важнейших операций манипулирования данными баз сетевого* типа можно отнести следующие: поиск записи в БД, переход от предка к первому потомку, переход от потомка к предку, создание новой записи, удаление текущей записи, обновление текущей записи, включение записи в связь, исключение записи из связи, изменение связей и т.д.

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. В

сравнении с иерархической моделью сетевая представляет большие возможности в смысле допустимости образования произвольных связей.

Недостатком сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе, а также сложность для понимания и выполнения обработки информации в БД обычным пользователем. Кроме того, в сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями. Поэтому системы на основе сетевой модели не получили широкого распространения на практике.

4.4 Реляционная модель

Довольно скоро на смену иерархической и сетевой моделям пришла принципиально новая модель данных – ***реляционная***. Она основана на принципе выявления подлежащих описанию в БД сущностей и связей между ними [16].

Сущность – это отдельный элемент деятельности организации (сотрудник или клиент, место или вещь, понятие или событие), который должен быть представлен в базе данных.

Атрибут – это свойство, которое описывает некоторый аспект объекта и значение которого следует зафиксировать.

Связь — это ассоциативное отношение между сущностями.

Реляционная модель данных обеспечивает возможности, которые делают управление БД и их использование относительно легким, устойчивым по отношению к ошибкам и предсказуемым:

– описывает данные с их естественной структурой, не добавляя каких-либо дополнительных структур, необходимых для машинного представления или для целей реализации;

– обеспечивает математическую основу для интерпретации выводимости, избыточности и непротиворечивости отношений;

– обеспечивает независимость данных от их физического представления, связей между данными и соображений реализации, связанных с эффективностью и подобными заботами.

Существенный вклад в развитие теории реляционных БД внес ученый Питер Чен Пин-Шен, который дополнил теорию моделирования данных доктора Э. Кодда весьма удобным описанием модели, получившей название ER-модель. Он предложил простой и эффективный способ представления сущностей, атрибутов и связей между ними в виде наглядных диаграмм на концептуальном уровне проектирования, то есть ER-модель отражает логическую природу представления данных.

4.5 Объектно-ориентированная модель

В середине 1980-х годов на рынке программных продуктов стали активно появляться программные средства, построенные на основе объектно-ориентированной парадигмы, основанной на принципах абстрагирования, инкапсуляции, модульности, иерархичности [17, 18].

Объектно-ориентированная база данных – база данных, в которой данные оформлены в виде моделей объектов, включающих прикладные программы, которые управляются внешними событиями. Результатом совмещения возможностей баз данных и возможностей объектно-ориентированных языков программирования являются ***объектно-ориентированные системы управления базами данных.*** Она позволяет работать с объектами баз данных так же, как с объектами в программировании на объектно-ориентированных языках программирования.

Причиной появления систем объектно-ориентированных баз данных была потребность в более адекватном представлении и моделировании сущностей реального мира, поскольку ООБД обеспечивают гораздо более развитую модель данных, нежели традиционные – реляционные базы данных.

Парадигма ООБД основывается на ряде базовых понятий: объект, идентифицируемость, класс, наследование, перегрузка, отложенное связывание.

В объектно-ориентированной модели данных любая сущность реального мира представляется всего одним понятием – ***объектом.*** С объектом ассоциируется состояние и поведение.

Состояние объекта определяется значениями его свойств (атрибутов), которыми могут являться примитивные значения (строки или целые числа) и непримитивные объекты. Непримитивный объект, в свою очередь, состоит из набора свойств. Следовательно, объекты можно рекурсивно определять в терминах других объектов.

Поведение объекта определяется с помощью методов, которые оперируют над состоянием объекта.

У каждого объекта имеется определяемый системой уникальный ***идентификатор.***

Объекты, обладающие одними и теми же свойствами и поведением, группируются в ***классы***. Объект может быть экземпляром одного класса или нескольких классов. Классы организуются в *иерархии классов*. *Подкласс* наследует свойства и методы суперкласса; кроме того, подклассы могут обладать индивидуальными свойствами и методами. В некоторых системах у класса может быть более одного суперкласса (множественное наследование), тогда как в других системах число суперклассов ограничено одним (одиночное наследование).

В большинстве моделей допускается перегрузка унаследованных свойств и методов. Перегрузка состоит в замене домена свойств новым доменом или в замене одной реализации метода другой его реализацией.

Объектно-ориентированные БД обладают рядом ***достоинств:***

– высокая точность моделирования реального мира;

–позволяют пользователям определять абстракции;

– облегчают проектирование некоторых связей;

– устраняют потребность в определяемых пользователями ключах;

– поддерживают новый набор предикатов сравнения;

–обеспечивают более высокую производительность, чем системы, основанные на реляционной модели;

– обеспечивают поддержку версий и длительных транзакций.

Однако, ООБД не лишены и ряда ***недостатков:***

– отсутствие привычных пользователю базовых средств СУБД,

– минимальная оптимизация и стандартная алгебра запросов;

– ограниченная поддержка ограничений целостности;

– ограниченная интеграция с существующими объектно-ориентированными системами программирования.

– отсутствие развитого математического аппарата, способного описывать данные в формате, приемлемом для объектно-ориентированной теории.

4.6 Слабоструктурированные данные

В тех ситуациях, когда на первое место выступает не столько решение задачи хранения записей, сколько простота организации и универсальность обмена данными, стоит обратить внимание на идею работы со слабоструктурированными данными. В качестве естественного способа хранения подобных данных выступает обычный текстовый формат.

Для обслуживания слабоструктурированных данных в 1998 г. разработан стандарт – ***расширяемый язык разметки*** (eXtensible Markup Language, XML). Необходимо отметить, что слабоструктурированные данные в формате XML не нуждаются в услугах отдельных СУБД, а для обработки XML разработан специальный прикладной интерфейс программирования [19, 20].

4.7 Документ-ориентированная модель

Документ-ориентированная (document-oriented) модель данных появилась в начале 2000-х. и выступает едва ли не антиподом реляционной теории. Она

опирается на идею хранения иерархических структур данных, в которой каждый элемент представляет собой целостный, а не разъединенный по двумерным отношениям, как в реляционной модели, документ. Документы хранятся не в таблицах, а в специальных хранилищах документов (document store).

Основным инструментом по доступу и управлению данными является декларативный язык NoSQL (Not only SQL).

Документ-ориентированные СУБД свободны от ключевого недостатка реляционных моделей – существенных затрат на сборку данных из нескольких таблиц в единое целое. В этом просто нет необходимости, ведь документ не разделен.

Однако, недостатком данной модели является ограниченность синтаксических конструкций по управлению данными языка NoSQL [5].

Контрольные вопросы:

1. Проанализируйте ключевые этапы развития моделей данных.
2. Обоснуйте значимость стандартизации в вопросе хранения данных.
3. Перечислите достоинства и недостатки иерархической модели данных.
4. Проанализируйте сильные и слабые стороны сетевой модели данных.
5. Каковы отличительные особенности реляционной модели данных?
6. Для чего нужны слабоструктурированные данные?
7. Что поставлено в основу объектно-ориентированной модели данных?
8. Каковы основные принципы построения объектно-ориентированной базы данных?
9. Проанализируйте сильные и слабые стороны объектно-ориентированной модели данных.
10. В чем заключается ключевое отличие в подходе хранения данных между реляционной и документ-ориентированной моделями?

ТЕМА 5 РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

Создателем реляционной модели считается математик Эдгар Фрэнк Кодд (Edgar Frank Codd, 1923–2003). Датой рождения реляционной теории можно считать июнь 1970 года. Именно тогда Кодд (на тот момент времени сотрудник одной из лабораторий корпорации IBM) опубликовал статью «Реляционная модель данных для больших совместно используемых банков данных», в которой впервые прозвучал столь популярный сегодня термин «реляционная модель» [10]. Строгое изложение теории реляционных баз данных (реляционной модели данных) в современном понимании было предложено Кристофером Дейтом [21].

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах, в которых каждая строка имеет один и тот же формат [4-8, 22].

Основными понятиями, с помощью которых определяется реляционная модель, являются следующие: отношение, схема отношения, домен, кортеж, кардинальность, атрибут, степень отношения, первичный ключ, значение атрибута, тип данных. Эти понятия имеют аналоги, соответствие которым приведено в табл. 1.

Таблица 1 – Форма представления элементов реляционной модели

Элемент реляционной модели	Форма представления (аналог)
Отношение	Таблица
Схема отношения	Строка заголовков таблицы
Кортеж (запись)	Строка таблицы
Атрибут (поле)	Столбец, заголовок столбца таблицы
Домен	Множество допустимых значений атрибута
Значение атрибута	Значение поля в строке
Кардинальность	Количество строк в таблице
Степень отношения	Количество столбцов в таблице
Первичный ключ	Уникальный идентификатор, один или несколько столбцов
Тип данных	Тип значений в полях таблицы

5.1 Сущность и атрибуты

Отношение – двумерная таблица, содержащая некоторые данные.

Сущность – реальный или абстрактный объект, данные о котором хранятся в одном или нескольких отношениях.

Атрибуты представляют собой свойства, характеризующие сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы.

Атрибуты могут быть простыми (например, название города, в который летит самолет) и *составными* (т. е. включать в себя несколько простых атрибутов). В качестве составного атрибута выступает адрес читателя. Здесь найдется место почтовому индексу, городу, улице, дому и номеру квартиры.

Атрибут может быть *производным*, то есть полученным в результате проведения какой-то операции над другими атрибутами (например значение взимаемого с сотрудников налога составляет определенный процент от заработной платы).

Можно столкнуться с *многозначными* атрибутами, они описывают те свойства сущности, в которых одновременно хранится несколько значений. Например, у одного человека может быть несколько контактных телефонных номеров.

Особая разновидность атрибутов, без которых невозможно построение реляционной модели, – это *атрибуты, однозначно идентифицирующие экземпляр сущности*. (например, индивидуальный номер налогоплательщика). Позднее подобные атрибуты превратятся в ***ключи*** реляционных таблиц.

Тип данных определяет особенности физического хранения данных.

Домен – это совокупность значений, из которых берутся значения соответствующих атрибутов определенного отношения. С точки зрения программирования домен – это тип данных, определяемый системой (стандартный) или пользователем.

Домен, опираясь на концепцию типа данных, несет дополнительную нагрузку – отвечает за поддержание логических правил описания данных (например, ввод корректной даты, проверка соответствия типа вводимых данных и ограничений поля). Благодаря доменным ограничениям в БД реализуется одно очень важное качество – доменная целостность данных.

Первичный ключ – это столбец или некоторое подмножество столбцов, которые уникально, т.е. единственным образом, определяют строки таблицы.

Первичный ключ, который включает более одного столбца, называют множественным, или комбинированным, или *составным.*

Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т.е. *не может иметь значение NULL.*

Если выбранный первичный ключ состоит из минимально необходимого набора атрибутов, то он является *не избыточным***.**

Ключи, которые можно также использовать в качестве первичных, называют возможными, потенциальными или альтернативными ключами.

Внешний ключ – это столбец или подмножество столбцов одной таблицы, которые могут служить в качестве первичного ключа для другой таблицы.

Внешний ключ является ссылкой на первичный ключ другой таблицы. Внешние ключи являются неотъемлемой частью реляционной модели, поскольку реализуют ***связи*** между таблицами базы данных.

Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных, называемое ссылочной целостностью. Это означает, что каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

На практике внешний ключ всегда будет составным, если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают. Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей.

Ключи обычно используют для достижения следующих целей:

– для исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются);

– упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним – возрастание, по другим – убывание);

– ускорения работы с кортежами отношения;

– организации связывания таблиц.

Отметим, что не любой таблице можно поставить в соответствие отношение. Условия, выполнение которых позволяет таблицу считать отношением, следующие:

– данные в ячейках таблицы должны быть структурно неделимыми (атомарными);

– данные в одном столбце должны быть одного типа;

– каждый столбец должен быть уникальным (недопустимо дублирование столбцов);

– столбцы и строки могут размещаться в таблице в произвольном порядке;

– столбцы имеют уникальные наименования.

К отношениям можно применять систему операций, позволяющую получать одни отношения из других. Например, результатом запроса к реляционной БД может быть новое отношение, вычисленное на основе имеющихся отношений. Основной единицей обработки данных в реляционных БД является отношение, а не отдельные его кортежи.

В реляционных СУБД установление связи между таблицами облегчает доступ к данным. Между таблицами могут устанавливаться бинарные (между двумя таблицами), тернарные (между тремя таблицами) и, в общем случае, n-арные связи.

Рассмотрим наиболее часто встречающиеся бинарные связи.

При связывании двух таблиц выделяют основную и дополнительную (подчиненную) таблицы. Логическое связывание таблиц производится с помощью ключа связи. Ключ связи, по аналогии с обычным ключом таблицы, состоит из одного или нескольких полей, которые в данном случае называют полями связи. Суть связывания состоит в установлении соответствия полей связи основной и дополнительной таблиц (рис. 19-20).

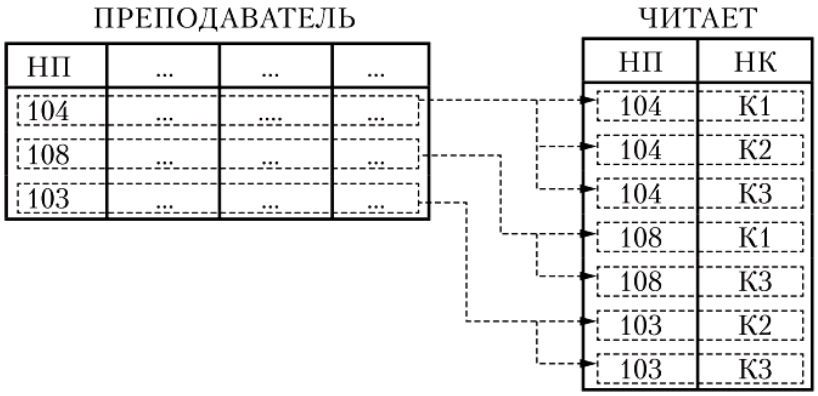


Рис. 19 – Связь главной таблицы ПРЕПОДАВАТЕЛЬ с подчиненной таблицей ЧИТАЕТ

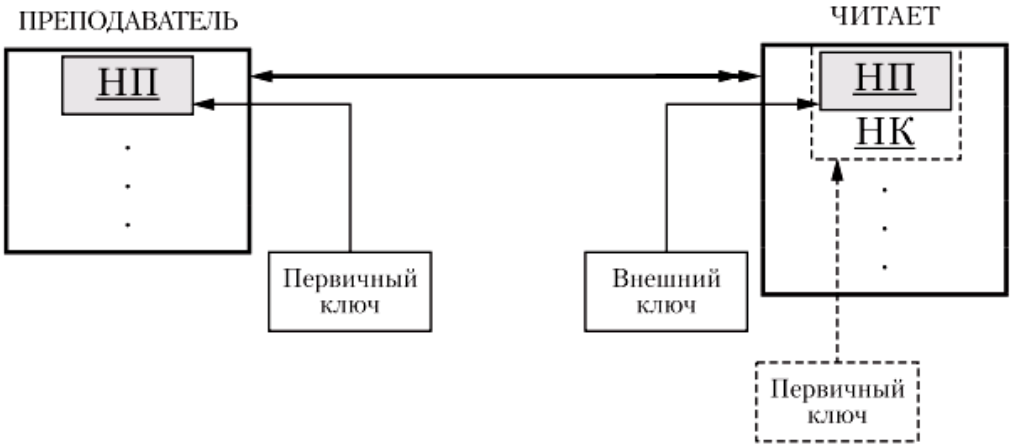


Рис. 20 – Схематическое изображение связи главной таблицы ПРЕПОДАВАТЕЛЬ с подчиненной таблицей ЧИТАЕТ

В зависимости от того, как определены поля связи основной и дополнительной таблиц между двумя таблицами могут устанавливаться следующие четыре основных вида связи:

– ***«один-к-одному»*** (1:1) – каждой записи из одной таблицы соответствует одна запись в другой таблице. Связь «один-к-одному» чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две;

– ***«один-ко-многим»*** (1:M) – каждой записи из одной таблицы соответствует несколько записей другой таблице. Левая сущность (со стороны «один») называется родительской, правая (со стороны «много») – дочерней;

– ***«многие-к-одному»*** (M:1) – множеству записей из одной таблице соответствует одна запись в другой таблице;

– ***«многие-ко-многим»*** – множеству записей из одной таблицы соответствует несколько записей в другой таблице. Тип связи «много-ко-многим» является временным типом связи, допустимым на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа «один-ко-многим» путем создания промежуточной сущности.

В целом концепция реляционной модели определяется двенадцатью ***правилами Кодда***:

1. Правило информации. Вся информация в базе данных должна быть представлена исключительно на логическом уровне и только одним способом – в виде значений, содержащихся в таблицах.

2. Правило гарантированного доступа. Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляционной базе данных должен обеспечиваться путем использования комбинации имени таблицы, первичного ключа и имени столбца.

3. Правило поддержки недействительных значений. В настоящей реляционной базе данных должна быть реализована поддержка недействительных значений, которые отличаются от строки символов нулевой длины, строки пробельных сим волов и от нуля или любого другого числа и

используются для представления отсутствующих данных независимо от типа этих данных.

4. Правило динамического каталога, основанного на реляционной модели. Описание базы данных на логическом уровне должно быть представлено в том же виде, что и основные данные, чтобы пользователи, обладающие соответствующими правами, могли работать с ним с помощью того же реляционного языка, который они применяют для работы с основными данными.

5. Правило исчерпывающего подъязыка данных. Реляционная система может поддерживать различные языки и режимы взаимодействия с пользователем (например, режим вопросов и ответов). Однако должен существовать, по крайней мере, один язык, операторы которого можно представить в виде строк символов в соответствии с некоторым четко определенным синтаксисом и который в полной мере поддерживает следующие элементы:

– определение данных;

– определение представлений;

– обработку данных (интерактивную и программную);

– условия целостности;

– идентификацию прав доступа;

– границы транзакций (начало, завершение и отмена).

6. Правило обновления представлений. Все представления, которые теоретически можно обновить, должны быть доступны для обновления.

7. Правило добавления, обновления и удаления. Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.

8. Правило независимости физических данных. Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним.

9. Правило независимости логических данных. Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.

10. Правило независимости условий целостности. Должна существовать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подъязыке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

11. Правило независимости распространения. Реляционная СУБД не должна зависеть от потребностей конкретного клиента.

12. Правило единственности. Если в реляционной системе есть низкоуровневый язык (обрабатывающий одну запись за один раз), то должна отсутствовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реляционном языке высокого уровня (обрабатывающем несколько записей за один раз).

5.2 Целостность данных

БД призвана на установленном разработчиком уровне абстракции отражать взаимоувязанные объекты реального мира. Чем точнее данное отражение, тем совершеннее база данных. Однако, помимо функции отражения предметной области, БД должна содержать некоторый набор правил, которые позаботятся о том, чтобы данные всегда находились в согласованном состоянии.

Целостность данных (data integrity) – соответствие значений всех данных базы данных определенному непротиворечивому набору правил.

Можно выделить три базовых класса правил, призванных поддерживать целостность данных в реляционной БД:

1) целостность доменов;

2) целостность сущностей;

3) ссылочная целостность.

Кроме того, существует понятие корпоративной целостности, это не что иное, как реализация в БД бизнес-правил, присущих конкретному предприятию.

Целостность доменов поддерживается за счет механизма доменных ограничений, т.е. описание домена включает некие логические правила, отбраковывающие некорректные значения, которые могут по ошибке попасть в атрибуты нашей таблицы. В простейшем случае допустимые значения могут быть просто перечислены, например «понедельник», «вторник» и т. д. Может быть объявлен диапазон допустимых значений, например от 2 до 5, если речь идет об оценках учеников. Все зависит от возможностей СУБД, в которой создается проект.

Особая роль в поддержании доменной целостности отводится особому определителю NULL. NULL обозначает *неопределенность или неизвестность.* При проектировании БД разработчики выявляют обязательные атрибуты, без которых работа БД невозможна, и атрибуты, к которым применяются пониженные требования, например которые разрешено заполнить позднее или не заполнять вовсе. Тогда пустая ячейка обозначает не нулевое значение атрибута, а пустое.

Благодаря определителю NULL базы данных работают в трехзначной логике. Поэтому в БД к известным каждому программисту значениям TRUE (истина) и FALSE (ложь) добавляется NULL (неопределенность).

Целостность сущностей направлена на обеспечение внутреннего единства отдельной сущности. Вне зависимости от содержания таблиц, следует соблюдать правило – *каждая строка таблицы обязана быть уникальной.* А для этого нужно контролировать корректность первичного ключа отношения. Суть требования к первичному ключу проста – во входящих в его состав атрибутах не должно содержаться ни одного определителя NULL.

Помимо поддержания корректности первичного ключа, целостность сущностей в состоянии обеспечить широкий спектр дополнительных сервисных возможностей. Их перечень определяется профессионализмом разработчика базы данных. Например, контроль непротиворечивости значения, вводимого в атрибут, на основе значений, имеющихся в других атрибутах таблицы или даже в атрибутах других таблиц. Так, БД железнодорожной кассы обязана уведомить оператора, что пассажир покупает на свое имя два билета в абсолютно противоположных направлениях на одно и то же время; БД в деканате не переведет на следующий курс студента, не сдавшего летнюю экзаменационную сессию.

Ссылочная целостность

Нарушение логических связей между таблицами может разрушить всю БД. Смысл правила заключается в следующем: внешнему ключу всегда должен соответствовать первичный ключ в главной таблице.

К нарушению связи могут привести следующие события (рассмотрим на примере – рис. 21).

Рис. 21 – Отношение «один ко многим»

Во-первых, удаление строки в главной таблице. Допустим, при удалении строки «Отдел кадров» в таблице «Отделы предприятия». в подчиненной таблице «Сотрудники» оказываются «брошенными» сотрудники Костенко и

Елецкова, ведь их внешний ключ по-прежнему будет хранить значение уже несуществующего первичного ключа.

Во-вторых, при добавлении новой записи в таблицу «Сотрудники» во внешнее поле «ключ_отдела» может попасть значение, которому нет соответствия в главной таблице.

В-третьих, ссылочная целостность может подвергнуться опасности при редактировании ключевых полей в главной или подчиненной таблице.

В современных СУБД все перечисленные проблемы предотвращаются автоматически, однако разработчику БД необходимо учитывать данное правило при разработке проекта.

5.3 Реляционная алгебра

Наилучшим способом описания реляционной модели данных является теория множеств. Э. Кодд предложил использовать восемь операций реляционной алгебры, определяющих особенности функционирования реляционной модели данных:

1) «σ» выборка (selection);

2) «↓» проекция (projection);

3) «×» декартово произведение (cartesian product);

4) «U» объединение, сложение (union);

5) «–» вычитание, разность (set difference);

6) «∩» пересечение (intersection);

7) «/» деление (division);

8) «||» соединение (join).

Реляционная алгебра представляет собой теоретический язык операций, которые на основе одного или нескольких отношений позволяют создавать другое отношение.

Операции выборки и проекции относятся к классу унарных, так как работают с одним отношением. При осуществлении ***выборки*** из исходного

множества формируется результирующее подмножество, содержащее только удовлетворяющие определенному условию элементы.

На языке множеств выборку можно сформулировать следующим образом. Пусть существует предикат выборки F, аргументами которого выступают атрибуты реляционной таблицы, состоящей из множества кортежей R. Результатом выборки окажется подмножество кортежей R', для которых предикат выборки F истинен:

$$\mathbf{R' = \sigma_F (R)}.$$

Например, с помощью операции выборки можем получить список студентов, родившихся в 2000 году (рис. 22). Выборка из таблицы «Студенты» составила три строки.

R - таблица "Студенты"

ID	SURNAME	FNAME	BIRTHDAY	SPECIALITY
1	Орехов	Владимир	11/04/1999	Математика
2	Петровский	Александр	21/11/2000	Математика
3	Кульгина	Оксана	5/01/2000	Ин. язык
4	Самойлов	Евгений	15/07/2001	Информатика
5	Кузьмина	Ирина	2/03/2000	Физика
6	Яковенко	Константин	12/09/2001	Химия
7	Ищенко	Владимир	09/19/2002	Астрономия

R' - выборка из таблицы "Студенты"

2	Петровский	Александр	21/11/2000	Математика
3	Кульгина	Оксана	5/01/2000	Ин. язык
5	Кузьмина	Ирина	2/03/2000	Физика

Рис. 22 – Демонстрация операции выборки

В отличие от операции выборки (выделяющей горизонтальные элементы исходного множества), ***операция проекции*** направлена на получение вертикальной составляющей множества.

Допустим, таблица R обладает n атрибутами, а нас интересует подмножество атрибутов с именами i_1, ..., i_k, причем $k \leq n$. Тогда результатом проекции станет: $\underset{i_1,\ldots,i_k}{\downarrow} R$.

На этот раз возвращаются все строки из исходной таблицы R, но в новое подмножество войдут не все, а только определенные пользователем столбцы таблицы.

Иллюстрацией операции может стать рисунок 23, на котором представлена проекция, полученная из таблицы студентов. В итоговое подмножество включены атрибуты с фамилией и именем студентов.

R - таблица “Студенты”

ID	SURNAME	FNAME	BIRTHDAY	SPECIALITY
1	Орехов	Владимир	11/04/1999	Математика
2	Петровский	Александр	21/11/2000	Математика
3	Кульгина	Оксана	5/01/2000	Ин. язык
4	Самойлов	Евгений	15/07/2001	Информатика
5	Кузьмина	Ирина	2/03/2000	Физика
6	Яковенко	Константин	12/09/2001	Химия
7	Ищенко	Владимир	09/19/2002	Астрономия

Проекция

Орехов	Владимир
Петровский	Александр
Кульгина	Оксана
Самойлов	Евгений
Кузьмина	Ирина
Яковенко	Константин
Ищенко	Владимир

Рис. 23 – Демонстрация операции проекции

Декартово произведение открывает набор операций, осуществляемых с двумя множествами кортежей R и S: **R × S.**

Отношение, получаемое в результате декартова произведения, представляет собой сцепление строк из одного отношения со строками из другого отношения. В результате произведения каждой строке из одной таблицы R присоединяется одна строка из другой таблицы S. На практике подобная операция редко когда оказывается востребованной, ведь в результирующем множестве оказываются все возможные сочетания строк из двух таблиц. Однако из любого правила есть и исключения. Допустим, что студенты хотят получить данные о том, какие учебные дисциплины им предстоит изучать (рис. 24).

R - “Студенты”

SURNAME	FNAME
Орехов	Владимир
Петровский	Александр
Кульгина	Оксана

X

S – “Дисциплины”

DISCIPLINE
Операционные системы
Языки программирования
Методы программирования
СУБД

R X S

SURNAME	FNAME	DISCIPLINE
Орехов	Владимир	Операционные системы
Орехов	Владимир	Языки программирования
Орехов	Владимир	Методы программирования
Орехов	Владимир	СУБД
Петровский	Александр	Операционные системы
Петровский	Александр	Языки программирования
Петровский	Александр	Методы программирования
Петровский	Александр	СУБД
Кульгина	Оксана	Операционные системы
Кульгина	Оксана	Языки программирования
Кульгина	Оксана	Методы программирования
Кульгина	Оксана	СУБД

Рис. 24 – Демонстрация операции произведения

Операция объединения проводится только с таблицами R и S, совместимыми по объединению, например с идентичной структурой. В результате мы получим суммарную таблицу, содержащую строки из двух исходных отношений: **R U S** (рис. 25).

R – “Студенты 1"

SURNAME	FNAME
Орехов	Владимир
Петровский	Александр
Кульгина	Оксана

U

S – “Студенты 2"

SURNAME	FNAME
Алфёров	Николай
Остапенко	Владимир
Юров	Олег

R ∪ S

SURNAME	FNAME
Орехов	Владимир
Петровский	Александр
Кульгина	Оксана
Алфёров	Николай
Остапенко	Владимир
Юров	Олег

Рис. 25 – Демонстрация операции объединения

Представленная на рисунке 25 операция объединения суммирует записи из таблиц R и S в результирующее отношение. Необходимо отметить, что если бы исходные таблицы содержали одинаковые по содержанию строки, то в

результате операции объединения из итогового множества должны быть удалены записи-дубликаты.

Операция вычитания позволяет выяснить, какие из строк первой таблицы отсутствуют во второй таблице. Как и в случае сложения, разность работает только с отношениями, совместимыми по объединению (с одинаковой структурой). Предположим, что у нас есть две таблицы R и S, в первой находятся данные о студентах учебной группы, а во второй – данные о студентах, успешно сдавших экзаменационную сессию.

Операция вычитания **R – S** позволит вычислить, кому следует явиться на пересдачу (рис. 26).

SURNAME	FNAME
Орехов	Владимир
Петровский	Александр
Кульгина	Оксана
Самойлов	Евгений
Кузьмина	Ирина
Яковенко	Константин
Ищенко	Владимир
Алфёров	Николай

–

SURNAME	FNAME
Орехов	Владимир
Петровский	Александр
Кульгина	Оксана
Самойлов	Евгений
Кузьмина	Ирина
Ищенко	Владимир
Алфёров	Николай

SURNAME	FNAME
Яковенко	Константин

Рис. 26 – Демонстрация операции вычитания

Операция пересечения может проводиться с таблицами, совместимыми по объединению, действие позволяет выявить строки, общие для двух таблиц. Допустим, что нам следует узнать, какие из студентов успешно сдали экзамен по высшей математике (таблица R) и экзамен по СУБД (таблица S). Пересечение двух отношений **R ∩ S** предоставит нам запрашиваемый результат (рис. 27).

R – “Студенты 1"

SURNAME	FNAME
Орехов	Владимир
Петровский	Александр
Кульгина	Оксана
Самойлов	Евгений
Кузьмина	Ирина
Яковенко	Константин
Щукин	Александр
Алфёров	Николай
Остапенко	Владимир
Юров	Олег

∩

S – “Студенты 2"

SURNAME	FNAME
Бобров	Валентин
Петровский	Александр
Кульгина	Оксана
Самойлов	Евгений
Кузьмина	Ирина
Ищенко	Владимир
Алфёров	Николай
Остапенко	Владимир

R ∩ S

SURNAME	FNAME
Петровский	Александр
Кульгина	Оксана
Самойлов	Евгений
Кузьмина	Ирина
Алфёров	Николай
Остапенко	Владимир

Рис. 27 – Демонстрация операции пересечения

Для понимания сути ***операции деления*** воспользуемся следующим примером. Допустим, в таблице R содержатся данные о том, какие виды занятий определенных дисциплин ведут педагоги университета. Нам необходимо уточнить, кто ведет лабораторные работы по СУБД и практические занятия по языкам программирования. Для этого в таблицу S заносятся две строки с указанными данными (рис. 28) и осуществляется деление: **P = R/S**.

R - делимое

SURNAME	DISCIPLINE	KIND
Орлов	СУБД	Лекция
Орлов	СУБД	Лаб. работа
Орлов	Языки программирования	Практ. занятие
Володина	СУБД	Лаб. работа
Володина	Языки программирования	Практ. занятие
Шуверов	Языки программирования	Практ. занятие

/

S - делитель

DISCIPLINE	KIND
СУБД	Лаб. работа
Языки программирования	Практ. занятие

P

SURNAME
Орлов
Володина

Рис. 28 – Демонстрация операции деления

Для того чтобы усвоить, как работает операция деления, надо запомнить две вещи:

– во-первых, результат деления исходной таблицы R на таблицу-делитель S будет содержать столбцы, отсутствующие в делителе (в нашем случае это атрибут фамилии «SURNAME»);

– во-вторых, в качестве строк в результат P войдут только те записи из делителя S, что при декартовом произведении результата на делитель P × S содержатся в делимом R.

Таким образом, после деления мы получаем две строки с фамилиями Орлов и Володина. Преподаватель Шуверов не попал в результирующее отношение, так как он не ведет лабораторные по дисциплине СУБД.

Операция соединения. Существует несколько способов соединения отношений (естественное соединение, внешнее соединение, тета-соединение и т. д.), и все они основаны на декартовом произведении, только теперь после перемножения строк двух таблиц на результат накладываются различные дополнительные ограничения. Допустим, в нашем распоряжении имеются две таблицы, в таблице S хранится информация о кафедрах вуза, а в таблице R – информация о преподавателях. Продемонстрируем наиболее распространенную операцию естественного соединения: **R ⊳⊲ S.**

Таблицы соединяются по общему столбцу, в таблице кафедр R это первичный ключ CHAIR_ID, а в таблице педагогов S это одноименный столбец внешнего ключа (рис. 29).

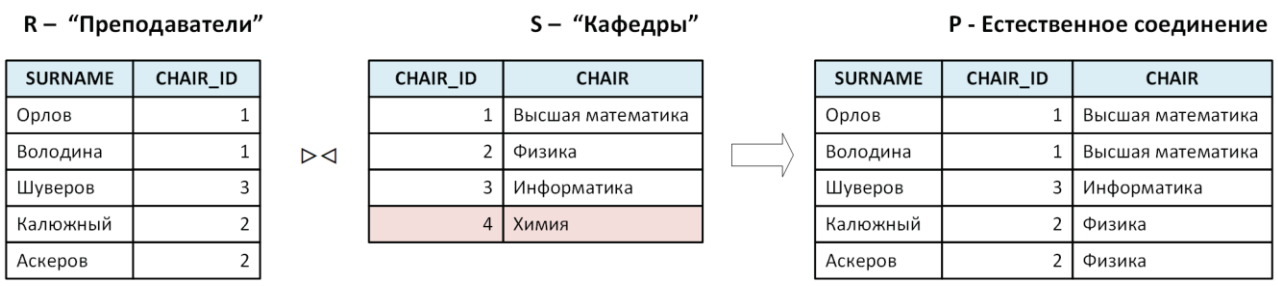


Рис. 29 – Операция естественного соединения

В результирующей таблице не нашлось места для кафедры химии. Это особенность естественного слияния, в таблице R нет ни одной записи с внешним ключом, ссылающимся на кафедру химии (CHAR_ID=4), поэтому кафедра химии не попала в итоговую таблицу. Если же нам требуется увидеть и «брошенные» кортежи, то вместо услуг естественного слияния необходимо воспользоваться помощью внешнего соединения.

Различают правое и левое внешние соединения. На рисунке 30 представлено правое внешнее соединение. Правым внешним соединением называется соединение, при котором кортежи отношения кафедр S (таблица расположена справа), не имеющие совпадающих значений в общих столбцах отношения преподавателей R, также включаются в результирующее отношение.

Рис. 30 – Демонстрация правого внешнего соединения

В одной из строк результирующей таблицы в столбце «SURNAME» мы обнаружим определитель NULL. Это произошло из-за того, что в таблице S педагогов не хранится информация о сотрудниках кафедры химии.

Тета-соединение определяет отношение, содержащее строки из декартового произведения отношений R и S, удовлетворяющие предикату F: $\mathbf{R \rhd\lhd_F S = \sigma_F(R \times S)}$.

При этом предикат F может использовать не только оператор равенства = (в этом случае тета-соединение превращается в уже знакомое нам естественное соединение), но и любой другой оператор сравнения ($<, \leq, \geq, >, \neq$).

Контрольные вопросы:

1. Что в реляционных БД понимается под типом сущности и сущностью?

2. Какие связи между сущностями могут быть отражены реляционной моделью?

3. Для чего предназначены доменные ограничения?

4. Для чего предназначен первичный ключ?

5. Какие требования предъявляются к реляционной таблице?

6. Каким образом организуется связь между двумя отношениями?

7. Как классифицируются связи между отношениями?

8. Что в реляционной модели понимается под целостностью данных?

9. Какие виды целостности данных вам известны?

10. Какие операции реляционной алгебры вам известны?

ТЕМА 6 КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ И ER-МОДЕЛЬ

На сегодняшний момент времени существует несколько эффективных решений для реализации концептуального проектирования БД, ключевым из которых является высокоуровневая концептуальная модель, разработанная Питером Ченом (Peter Pin-Shan Chen). Указанная модель известна под названием ***«сущность-связь»*** (Entity-Relationship), или просто ***ER-модель.***

Впервые она была анонсирована в марте 1976-го, тогда П. Чен опубликовал работу «The Entity-Relationship Model. Toward a Unified View of Data» в научном журнале «Transactions on Database Systems (TODS)».

Модель «сущность-связь» относится к разряду концептуальных. Другими словами, она представляет общий взгляд на данные и позволяет на понятийном уровне разобраться с тем, что будет представлено в будущей базе данных.

Достоинство предложенного П. Ченом решения заключается в том, что ER-модель представляется в виде наглядных графических диаграмм. Терминология ER-модели опирается понятия «тип сущности», «атрибут» и «связь».

6.1 Типы сущностей и атрибуты

Построение ER-модели начинается с выявления всех типов сущностей, подлежащих хранению в БД. Различают две разновидности типов сущностей: *слабую и сильную*. Слабый тип находится в зависимом состоянии от сильной сущности, напротив, сильный тип вполне самостоятелен и ни от кого (или чего) не зависит. На диаграммах сильный тип изображается в виде прямоугольника с названием типа сущности внутри него, для обозначения слабого типа сущности контур прямоугольника рисуется двойной линией (рис. 31).

Каждый тип сущности обладает некоторым ***набором атрибутов***, в которых хранятся значения, описывающие конкретную сущность.

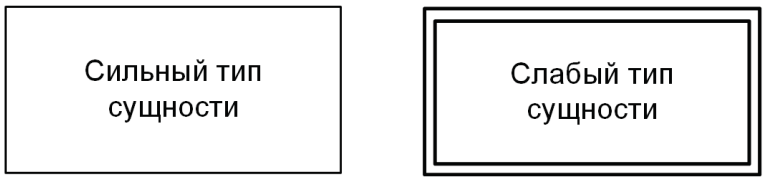


Рис. 31 – Обозначение слабого и сильного типов сущности

Различают *простые, составные, однозначные, многозначные и производные* атрибуты.

Главное отличие между простым и составным атрибутами в том, что простой состоит из одного компонента, а составной из нескольких. Примером составного атрибута может стать почтовый адрес, он включает группу простых атрибутов (индекс, страна, город, улица, дом).

Однозначный атрибут содержит одно-единственное значение для сущности, например фамилию для типа сущности «Сотрудник». Многозначный атрибут допускает одновременное хранение нескольких значений, например у сотрудника может быть несколько телефонных номеров.

Производный атрибут содержит значение, полученное на основе данных, хранящихся в других атрибутах. Например, возраст сотрудника можно легко вычислить, зная день его рождения и сегодняшнее число. Графическое представление атрибутов на ER-диаграмме представлено на рисунке 32.

В реляционных таблицах производные атрибуты редко когда превращаются в реальные столбцы таблицы, физически хранящие значения, так ка их можно получить путем элементарных вычислений.

Для изображения на схеме атрибута применяется эллипс, к своему типу сущности он присоединяется линией. Название атрибута записывается внутри эллипса. Если атрибут содержит идентификатор сущности (позднее он превратится в первичный ключ отношения), то название атрибута подчеркивается. Производный атрибут обводится пунктирной линией, многозначный – двойной.

Рис. 32 – Представление атрибутов на диаграммах ER-модели

При рисовании многозначного атрибута некоторые разработчики вместо черчения эллипса с двойным контуром рисуют обычный эллипс, но соединяют его с сущностью двойной линией. То же самое можно сказать и о производном атрибуте, только на этот раз эллипс соединяют с прямоугольником пунктирной линией.

Рассмотрим представленный на рисунке 33 фрагмент ER-модели.

Это диаграмма «Employee», описывающая тип сущности «сотрудник предприятия». На диаграмме представлены все имеющиеся разновидности атрибутов. Атрибут «Employee_id» выполняет функции первичного ключа сущности. Имя «FName» и фамилия «LName» являются ингредиентами составного атрибута «EmployeeName». Кроме этого, имеется еще один составной атрибут «Address», он хранит почтовый адрес служащего. Обведенный двойной линией многозначный атрибут «PhoneNum» говорит нам о том, что у реального человека может быть несколько контактных телефонных номеров.

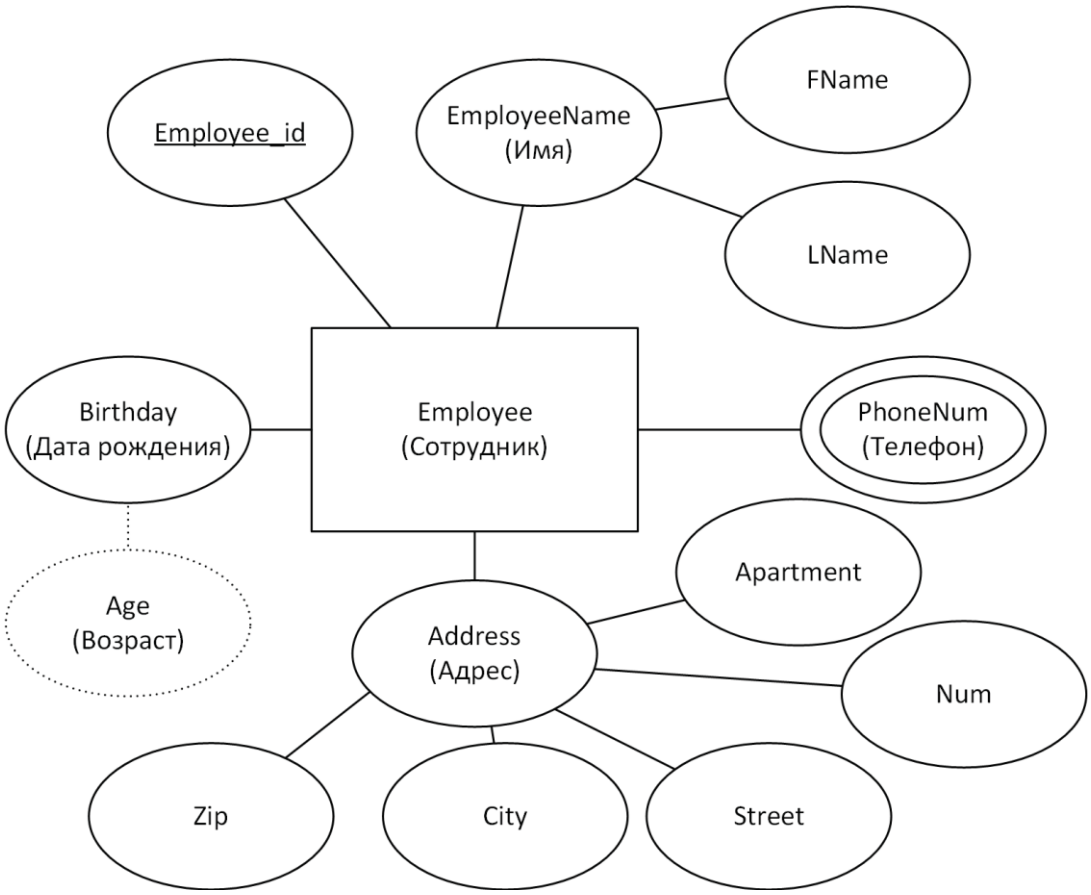


Рис. 33 – Представление типа сущности на диаграммах ER-модели

Производный атрибут «Age» отражает возраст сотрудника, на схеме этот атрибут соединен с датой рождения «Birthday». Такая подробность подскажет разработчику, что расчет возраста должен вестись от даты появления на свет работника предприятия.

6.2 Подтипы сущностей

Чем больше в организации числится персонала, тем сложнее организовать учет данных о нем. Люди обладают разными специальностями, профессиональными навыками, личностными характеристиками, которые необходимы и подлежат учету при выполнении различных работ на соответствующих должностях (научная степень, категория, разряд, спортивные достижения и т.д.).

Поэтому, кроме общих данных (фамилия и имя, дата рождения и т. п.), появляются уникальные черты, характерные только для узких профессиональных групп. Решением подобных задач может стать попытка определить один-единственный универсальный тип сущности (например, как на рис. 33) и, пытаясь учесть все возможные случаи жизни, снабдить его всем возможным спектром атрибутов. Решение вполне работоспособно, но абсолютно нерационально.

Решением данной проблемы являются *подтипы сущности.* Это тот случай, когда общая для всех сотрудников информация хранится в головном типе сущности (супертипе), а нюансы и тонкости выносятся в несколько специализированных подтипов.

В рассмотренном примере количество подтипов определяется числом профессиональных групп сотрудников предприятия (рис. 34).

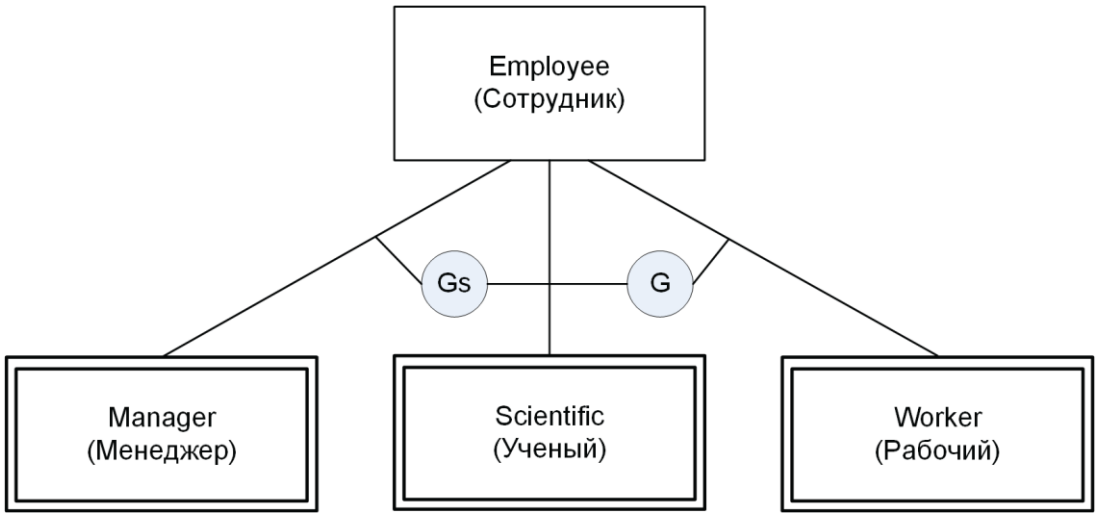


Рис. 34 – Изображение супертипа и подтипов на диаграмме

Все атрибуты супертипа в равной степени принадлежат всем подтипам, ведь у любого сотрудника есть имя и дата рождения. А информация, хранимая в подтипах сущности, специфична для каждой из групп.

Необходимо обратить внимание на то, что подтипы «Manager» и «Scientific» помечены кружочком с символами «Gs», а подтип «Worker»

отмечен символом «G». Символы «Gs» указывают на то, что подтипы относятся к разряду пересекающихся, т. е. ученый одновременно может быть и управленцем нашего предприятия. В таком случае мы допускаем, что данные об одном и том же человеке могут одновременно находиться в таблицах, учитывающих менеджеров и ученых. Символ «G» отмечает непересекающийся тип – рабочий не может одновременно являться ученым или менеджером.

Супертип и подтипы состоят друг с другом в отношениях «один к одному». Это едва ли не единственный случай в реляционных БД, когда появление связи 1:1 значительно упрощает процесс проектирования и сопровождения БД.

Однако, в процессе разработки проектировщик может столкнуться со следующими проблемами:

– не существует единого правила поддержания целостности данных при организации взаимодействия между головной и подчиненными таблицами;

– возникают определенные сложности в определении первичных ключей в таблицах подтипов;

– средствами современного SQL весьма сложно создать запрос, объединяющий всю информацию из таблицы-супертипа и всех таблиц подтипов;

– достаточно тяжело обеспечить безопасный доступ пользователя к столбцам таблиц подтипов.

Вместе с тем построенные на основе идеи подтипов сущностей базы данных приобретают следующие преимущества:

– устраняется избыточность данных, так как каждый подтип хранит только необходимую информацию;

– упрощается процесс определения доменных ограничений для столбцов таблиц подтипов;

– появление новой (не учтенной в момент проектирования БД) классификационной группы записей приводит к созданию очередной таблицы подтипов. Поэтому не возникает нужды реструктурировать таблицу супертипа

(добавление новых или изменение старых столбцов), с вытекающими отсюда проблемами кардинальной переработки всех клиентских приложений.

6.3 Связи в ER-модели

Сформировав полный перечень всех подлежащих учету типов сущностей и их атрибутов, создатель ER-модели переходит к выявлению ассоциаций между типами сущностей и построению связей между ними.

Для того чтобы разработчик мог на схеме отразить смысловую нагрузку связи между сущностями, показывающую характер взаимодействия между сущностями, она записывается внутри ромба. С целью упрощения диаграммы допускается опускать атрибуты типов сущностей, ограничиваясь только первичными ключами (рис. 35).

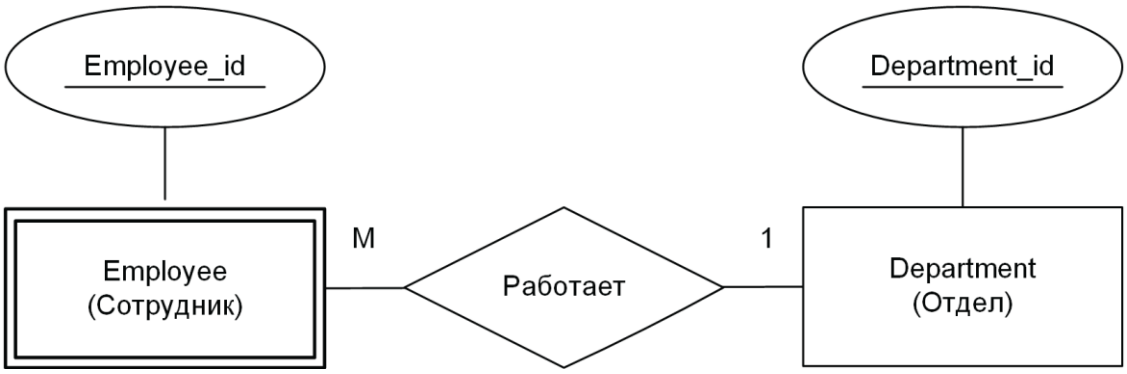


Рис. 35 – Представление связи 1:M на диаграммах ER-модели

В ER-моделировании различают три типа связи между типами сущностей: «один к одному» (1:1), «один ко многим» (1:M) и «многие ко многим» (M:N).

При появлении на диаграмме связи типа 1:1 следует проанализировать, не является ли предполагаемый тип сущности всего лишь атрибутом. Исключение составляет рассмотренный механизм супертипов и подтипов.

На рисунке 35 представлена наиболее часто встречающаяся связь «один ко многим» – в одном отделе работает много сотрудников. На диаграмме отдел «Department» представлен как сильная сущность, сотрудник «Employee» – как слабая. Такое решение объясняется тем, что сотрудник находится в подчиненном отношении к отделу, в котором он трудится.

Часто программисты для повышения информативности диаграмм при обозначении связи вместо использования символа «M» явным образом указывают мощность связи. Например, обозначение (1, 10) говорит о том, что минимальное значение мощности соответствует 1, а максимальное – 10 (другими словами, в отделах может работать от 1 до 10 сотрудников).

Проектируя ER-модель, следует особое внимание уделять связи «многие ко многим». Вполне реальна ситуация, когда несколько сотрудников одновременно выполняют несколько производственных поручений в заказе «Order» (рис. 36).

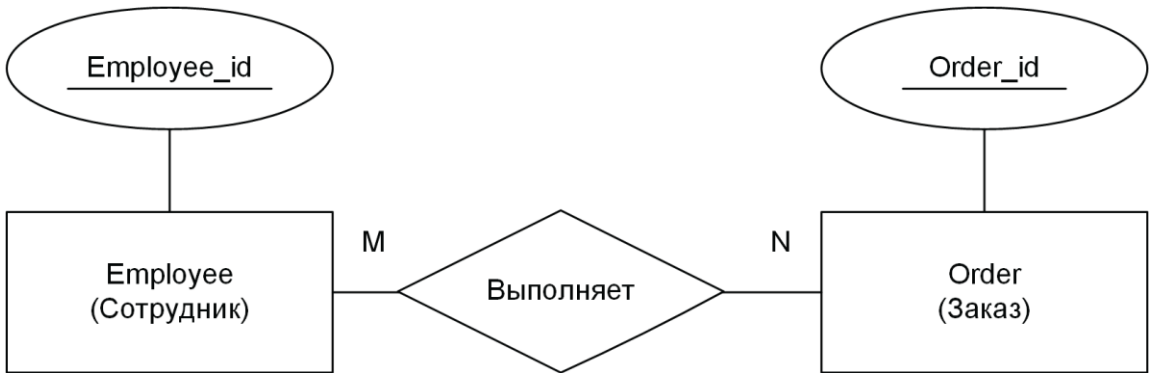


Рис. 36 – Представление связи M:N на диаграммах ER-модели

Например, один и тот же автомеханик в рабочий день может получить заявки на обслуживание нескольких автомобилей, при этом отдельно взятый автомобиль может попасть в руки нескольких специалистов (допустим, на ремонт двигателя и на замену электропроводки).

В тех случаях, когда между двумя типами сущностей возникает связь «многие ко многим», разработчик БД создает искусственный тип сущности, выполняющий функции коммутатора между основными сущностями (рис. 37).

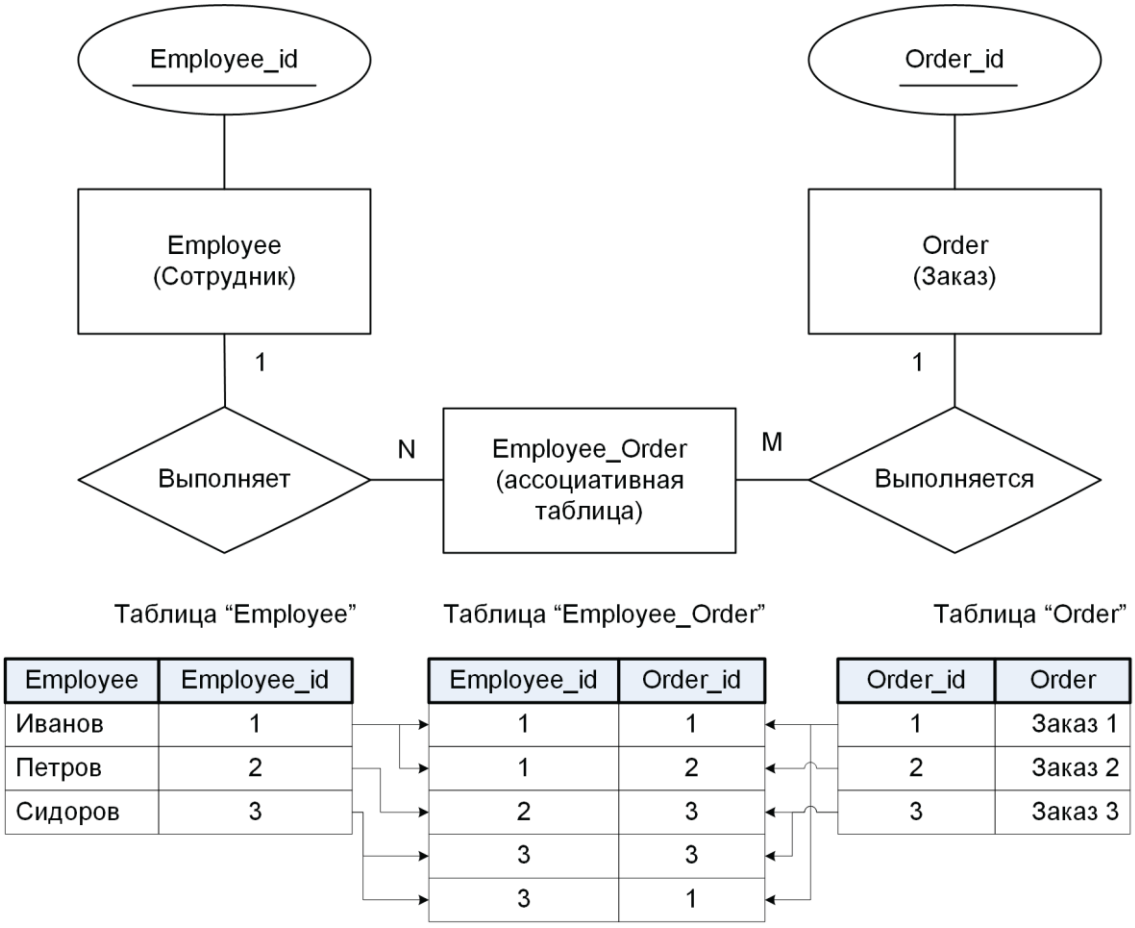


Рис. 37 – Преобразование отношения M:N к двум отношениям 1:M

Дополнительный тип сущности разрывает связь M:N, что позволяет трансформировать ее в пару обычных связей «один ко многим». Искусственно созданная сущность-коммутатор будет содержать два атрибута для внешних ключей, которые обеспечат связь между сущностями («Employee» и «Order»).

На диаграммах большинство разработчиков ER-модели используют компромиссное решение (рис. 38) – на диаграммах искусственный тип сущности изображается в виде ромба, вписанного в прямоугольник.

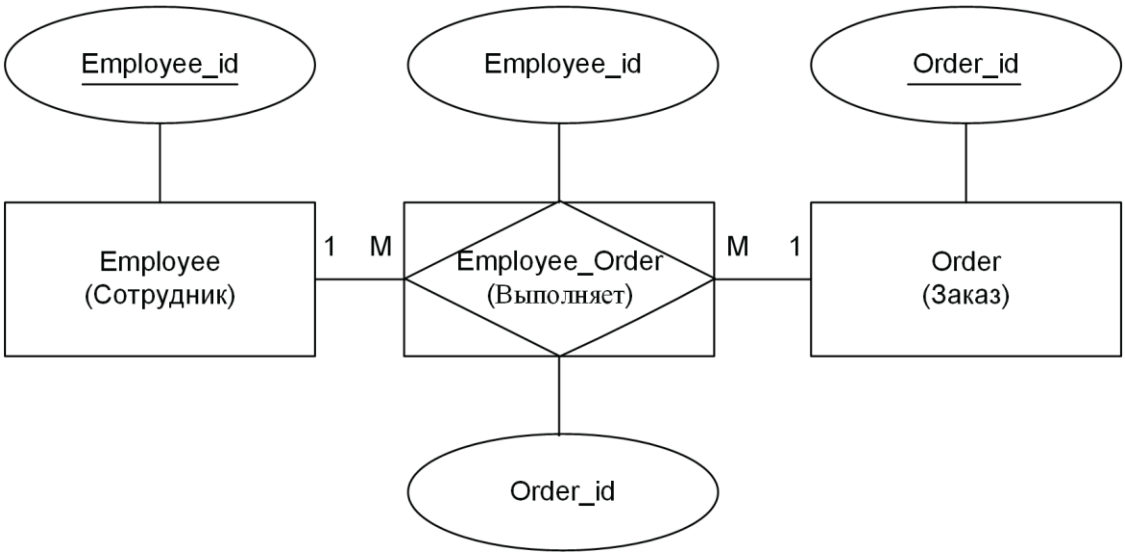


Рис. 38 – Коммутирующий тип сущности на ER-диаграмме

Такое графическое решение, с одной стороны, указывает на то, что это будущая таблица, а с другой – напоминает, что своим появлением таблица обязана необходимости организовать коммутацию между другими таблицами.

6.4 Сильные и слабые связи

Рассмотрим еще одну особенность взаимодействия между типами сущностей в БД – наличие сильных и слабых связей.

Сильная связь обычно возникает между сильным и слабым типами сущностей в тех условиях, когда существование слабого типа невозможно без поддержки сильного. Например, слабая сущность сотрудник не может трудиться на предприятии, не входя в штат какого-либо из отделов. Напротив, в той ситуации, когда присутствие связи между двумя типами сущности не обязательно, говорим о *слабой связи*. Слабые связи надо искать между типами сущностей, не находящимися в прямой зависимости друг от друга и, как следствие, способными существовать автономно. Например, работа над некоторыми заказами может осуществляться не только в стенах нашего предприятия, но и совместно с каким-то соисполнителем (смежной фирмой).

Несмотря на то что тип сущности «Accomplice» (соисполнитель) относится к разряду сильных, так как может существовать самостоятельно, связь между заказом и смежником оказывается слабой. Ведь предприятию не всегда нужны помощники, и большинство заказов оно выполняет своими силами. Об этом информирует маленький кружок со стороны необязательной сущности (рис. 39).

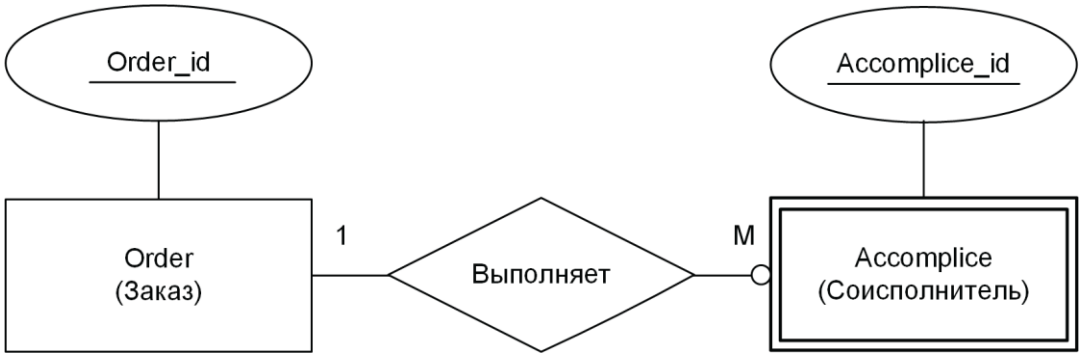


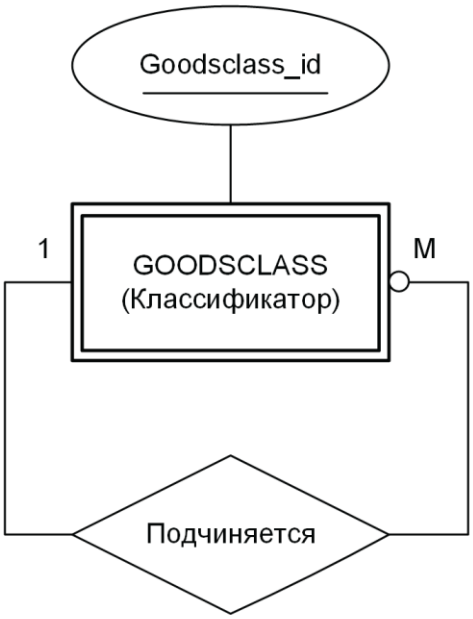
Рис. 39 – Пример необязательной связи

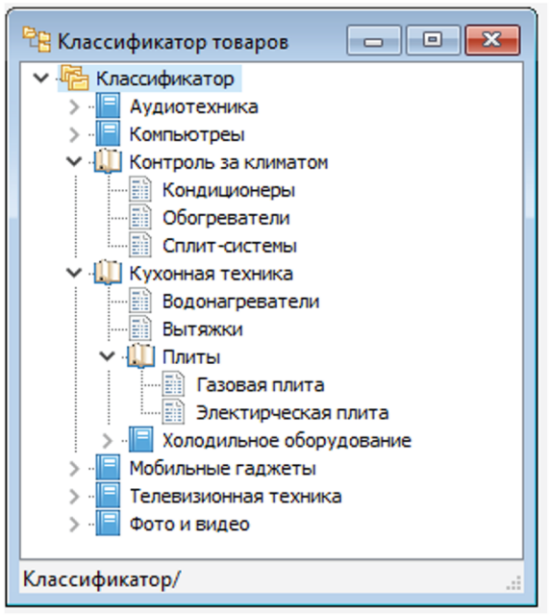
В последующем, на этапе создания реляционной таблицы «Order», кружок подскажет нам о том, что информация о соисполнителе заказа необязательна и поэтому поле внешнего ключа допускает вставку определителя NULL.

В некоторых нотациях ER-модели для обозначения сильной связи на диаграммах очерчивают ромб двойной линией.

6.5 Рекурсивная связь

Рассмотрим возможность создания унарных связях, которые эффективны при организации рекурсивных отношений внутри одной таблицы. Замыкание типа сущности на самого себя окажется весьма полезным при описании многоуровневых иерархических структур, подобных классификатору товаров, хранящихся на складе (рис. 40).





Фрагмент данных таблицы “GOODSCLASS”

Goodsclass_id	Parent_id	Goodsclass
1	NULL	Классификатор
2	1	Аудиотехника
3	1	Компьютеры
4	1	Контроль за климатом
5	4	Кондиционеры
6	4	Обогреватели
7	4	Сплит-системы

Рис. 40 – Представление унарной рекурсивной связи на диаграмме ER-модели

Здесь тип сущности «Goodsclass» содержит названия категорий товаров, которыми торгует автоматизируемая фирма. Между категориями явно просматривается правило взаимодействия главный–подчиненный.

На практике для построения иерархии реальной реляционной таблице потребуется как минимум три столбца: поле первичного ключа «Goodsclass_id», поле внешнего ключа «Parent_id» и содержащее названия категорий текстовое поле «Goodsclass».

Рекурсивная связь между записями таблицы обеспечивается за счет взаимодействия внешнего и первичного ключей, с этой целью поле внешнего

ключа дочернего элемента хранит значение первичного ключа родительского узла. Если же идет речь о самом старшем элементе иерархии, который никому не подчинен (папка «Классификатор»), то в его внешнем ключе окажется определитель NULL (рис. 40). Например, узлу «Контроль за климатом» принадлежат дочерние узлы «Кондиционеры», «Обогреватели» и «Сплит-системы». В полях внешнего ключа всех трех подчиненных узлов находим число 4, а это и есть значение первичного ключа родительского узла «Контроль за климатом».

6.6 Связи высокого порядка

Модель Питера Чена допускает возникновение связей более высокого порядка, чем бинарные: тернарных, кватернарных и т. д. Пример тернарной связи представлен на рисунке 41.

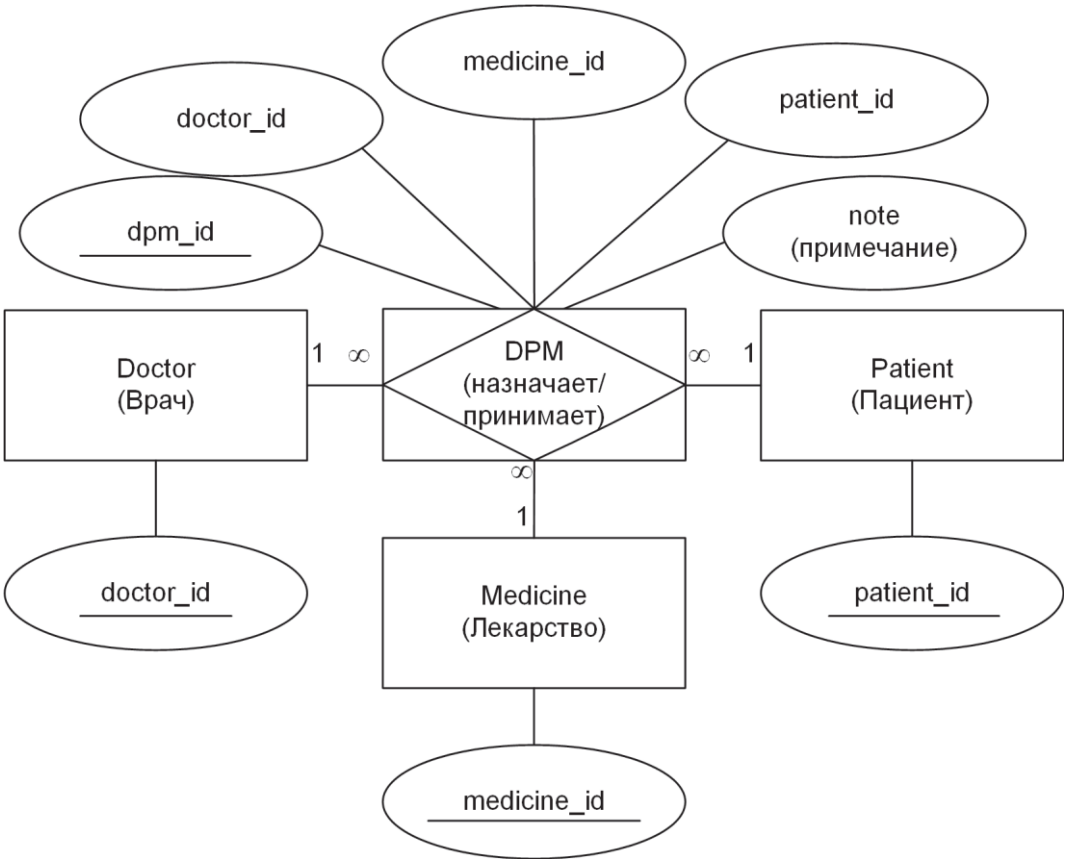


Рис. 41 – Тернарная связь между типами сущностей

В данном случае процесс проектирования практически такой же, как и в подробно рассмотренной выше (рис. 36 и 37) ситуации с поддержанием связи «многие ко многим» между двумя типами сущностей.

Задача решается путем введения искусственной сущности-коммутатора. Коммутатор «DPM» (название создано по первым буквам коммутируемых объектов) объединяет врачей, пациентов и назначаемые лекарства.

Помимо трех внешних ключей, предназначенных для организации связи с указанными сущностями, искусственная таблица снабжена первичным ключом и атрибутом «dpm_id». в котором будут хранится рецепты и назначения лекарств пациентам.

Контрольные вопросы:

1. Что понимается под концептуальным проектированием БД?

2. Какую роль в процессе проектирования базы данных играют высокоуровневые модели данных?

3. Назовите основные концепции ER-модели и укажите способ их представления на диаграммах Питера Чена.

4. Дайте классификацию атрибутов ER-модели.

5. Разъясните предназначение подтипов сущностей.

6. Какие типы связей поддерживает ER-модель?

7. Что такое «мощность» связи?

8. Проанализируйте процесс перехода от связи M:N к 1:M и отображение его в ER-модели.

9.Укажите назначение рекурсивной связи.

10. Проанализируйте построение ER-модели в случае наличия тернарной связи.

ТЕМА 7 ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ И НОРМАЛИЗАЦИЯ

Переход от этапа концептуального проектирования к логическому, связанному с непосредственной разработкой реляционных таблиц, должен сопровождаться нормализацией данных.

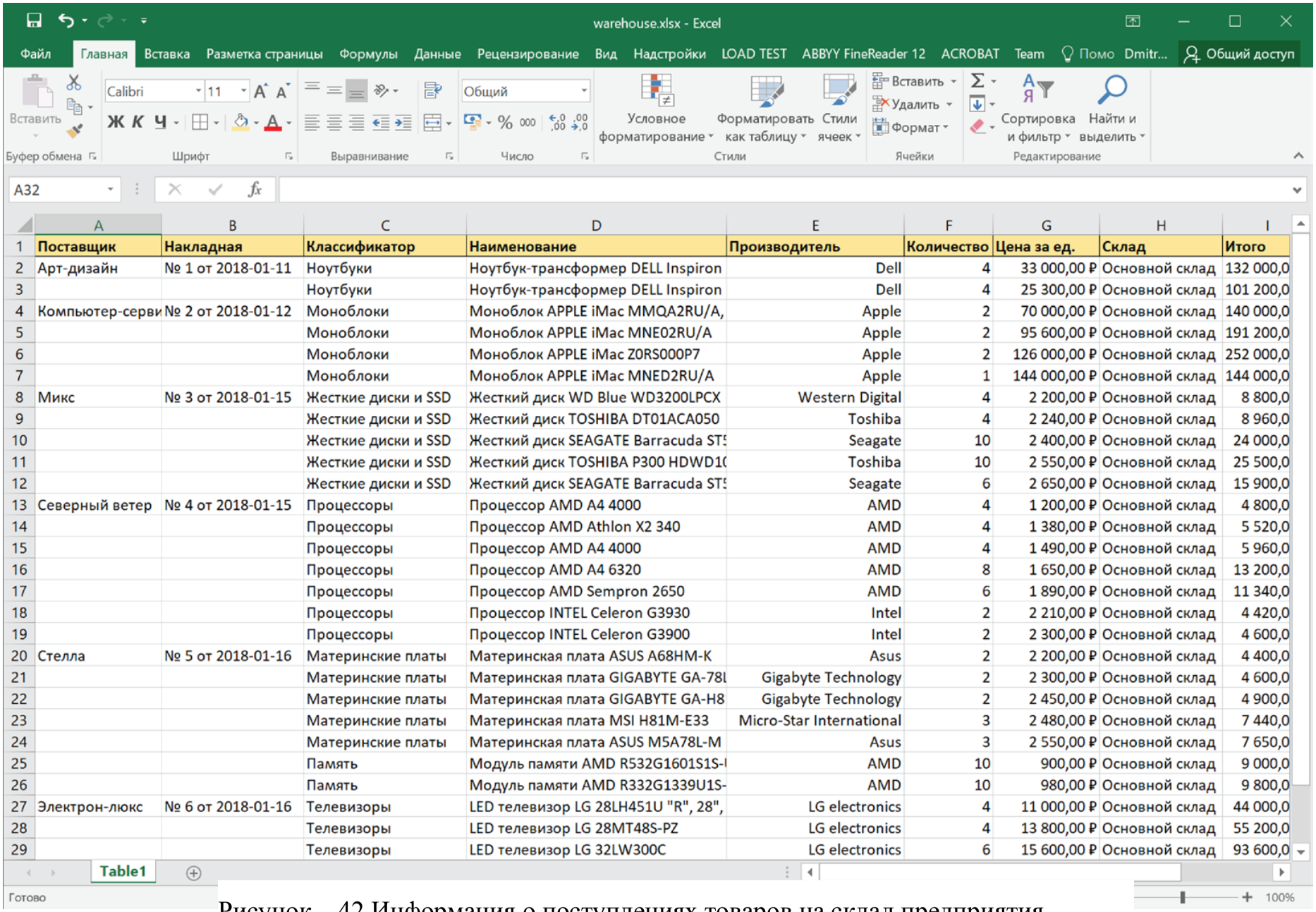
Нормализация преследует несколько целей. Наиболее важная из них – борьба с избыточностью данных. Устранение избыточности достигается не просто за счет рационального назначения размерности атрибутам таблиц, а за счет формирования эффективных взаимоувязанных табличных структур, объединяющих несколько отношений.

В результате нормализации БД таблицы приобретают дополнительную гибкость, что делает возможным изменение структуры таблиц в будущем, например, в процессе модернизации действующей БД.

Нормализация данных приводит не только к минимизации расходов на хранение данных, но и устраняет ряд косвенных проблем, точнее аномалий, присущих неграмотно построенным таблицам. К таковым относятся *аномалии вставки, редактирования и удаления данных*.

Рассмотрим фрагмент складской ведомости, содержащей информацию о поступлениях товаров на склад предприятия (рис. 42).

Для обычного покупателя или продавца вся совокупность собранной в таблице информации понятна и находит однозначное толкование. В результате первой поставки от компании «Арт-дизайн» на склад поступили ноутбуки компании Dell. Работник магазина сделал об этом соответствующую запись в книге учета. Все ноутбуки оприходованы по накладной № 1 от 11.01.2018. В поступлении имеется два типа ноутбуков: 4 шт. по цене 33 000 руб. и 4 шт. по цене 25 300 руб. При внесении второй строки работник не стал повторно указывать название фирмы-поставщика, номер и дату приходного ордера. Этим же правилом оператор руководствовался каждый раз, сталкиваясь с подобными повторами.



	A	B	C	D	E	F	G	H	I
1	Поставщик	Накладная	Классификатор	Наименование	Производитель	Количество	Цена за ед.	Склад	Итого
2	Арт-дизайн	№ 1 от 2018-01-11	Ноутбуки	Ноутбук-трансформер DELL Inspiron	Dell	4	33 000,00 ₽	Основной склад	132 000,0
3			Ноутбуки	Ноутбук-трансформер DELL Inspiron	Dell	4	25 300,00 ₽	Основной склад	101 200,0
4	Компьютер-серви	№ 2 от 2018-01-12	Моноблоки	Моноблок APPLE iMac MMQA2RU/A,	Apple	2	70 000,00 ₽	Основной склад	140 000,0
5			Моноблоки	Моноблок APPLE iMac MNE02RU/A	Apple	2	95 600,00 ₽	Основной склад	191 200,0
6			Моноблоки	Моноблок APPLE iMac Z0RS000P7	Apple	2	126 000,00 ₽	Основной склад	252 000,0
7			Моноблоки	Моноблок APPLE iMac MNED2RU/A	Apple	1	144 000,00 ₽	Основной склад	144 000,0
8	Микс	№ 3 от 2018-01-15	Жесткие диски и SSD	Жесткий диск WD Blue WD3200LPCX	Western Digital	4	2 200,00 ₽	Основной склад	8 800,0
9			Жесткие диски и SSD	Жесткий диск TOSHIBA DT01ACA050	Toshiba	4	2 240,00 ₽	Основной склад	8 960,0
10			Жесткие диски и SSD	Жесткий диск SEAGATE Barracuda ST5	Seagate	10	2 400,00 ₽	Основной склад	24 000,0
11			Жесткие диски и SSD	Жесткий диск TOSHIBA P300 HDWD1(	Toshiba	10	2 550,00 ₽	Основной склад	25 500,0
12			Жесткие диски и SSD	Жесткий диск SEAGATE Barracuda ST5	Seagate	6	2 650,00 ₽	Основной склад	15 900,0
13	Северный ветер	№ 4 от 2018-01-15	Процессоры	Процессор AMD A4 4000	AMD	4	1 200,00 ₽	Основной склад	4 800,0
14			Процессоры	Процессор AMD Athlon X2 340	AMD	4	1 380,00 ₽	Основной склад	5 520,0
15			Процессоры	Процессор AMD A4 4000	AMD	4	1 490,00 ₽	Основной склад	5 960,0
16			Процессоры	Процессор AMD A4 6320	AMD	8	1 650,00 ₽	Основной склад	13 200,0
17			Процессоры	Процессор AMD Sempron 2650	AMD	6	1 890,00 ₽	Основной склад	11 340,0
18			Процессоры	Процессор INTEL Celeron G3930	Intel	2	2 210,00 ₽	Основной склад	4 420,0
19			Процессоры	Процессор INTEL Celeron G3900	Intel	2	2 300,00 ₽	Основной склад	4 600,0
20	Стелла	№ 5 от 2018-01-16	Материнские платы	Материнская плата ASUS A68HM-K	Asus	2	2 200,00 ₽	Основной склад	4 400,0
21			Материнские платы	Материнская плата GIGABYTE GA-78L	Gigabyte Technology	2	2 300,00 ₽	Основной склад	4 600,0
22			Материнские платы	Материнская плата GIGABYTE GA-H8	Gigabyte Technology	2	2 450,00 ₽	Основной склад	4 900,0
23			Материнские платы	Материнская плата MSI H81M-E33	Micro-Star International	3	2 480,00 ₽	Основной склад	7 440,0
24			Материнские платы	Материнская плата ASUS M5A78L-M	Asus	3	2 550,00 ₽	Основной склад	7 650,0
25			Память	Модуль памяти AMD R532G1601S1S-I	AMD	10	900,00 ₽	Основной склад	9 000,0
26			Память	Модуль памяти AMD R332G1339U1S-	AMD	10	980,00 ₽	Основной склад	9 800,0
27	Электрон-люкс	№ 6 от 2018-01-16	Телевизоры	LED телевизор LG 28LH451U "R", 28",	LG electronics	4	11 000,00 ₽	Основной склад	44 000,0
28			Телевизоры	LED телевизор LG 28MT48S-PZ	LG electronics	4	13 800,00 ₽	Основной склад	55 200,0
29			Телевизоры	LED телевизор LG 32LW300C	LG electronics	6	15 600,00 ₽	Основной склад	93 600,0

Рисунок – 42 Информация о поступлениях товаров на склад предприятия

Предложенный на рисунке 42 подход к обработке информации вполне приемлем для небольшого магазина, владелец которого сам выбрал способ учета товара. Но, в отличие от человека, компьютер требует более формализованного подхода. Вполне вероятно, что компьютер не сможет правильно сопоставить строку товара с накладной из-за незаполненных ячеек таблицы. При построении фильтра по столбцу компьютер однозначно ошибется, если при введении классификатора сначала напишем «Материнские платы», затем «Motherboard», а третий раз вообще остановимся на сокращении «МП». Оператор ПК понимает, что речь идет об одном и том же, а для компьютера это не так.

Поэтому, вполне понятная с нашей точки зрения информация для ПК может оказаться некорректной. Это приводит к *аномалии вставки*.

Аномалию удаления рассмотрим на примере имени поставщика и номера приходной накладной (рис. 42). Обратим внимание, что большинство поставок не ограничивалось одной строкой и включало две и более позиций. Так, первому в списке приходному ордеру № 1 на склад поступило два наименования товара, по № 2 – 4 и т. д. Допустим, что в ордер № 3 ошибочно внесен жесткий диск «WD Blue WD3200LPCX» и необходимо удалить связанную с ним строку из таблицы. Вместе с диском мы рискуем удалить и все данные о номере и дате приходного ордера. В результате оставшиеся строки накладной просто потеряют связь с поставщиком и накладной, что приведет к нарушениям в складском учете. Соответственно, при удалении строки с данными о поставщике и ордере потребуется перенести эти сведения в следующую за удаляемой строку.

Также в примере встречаются повторяющиеся данные (например, в столбцах «Поставщик», «Классификатор» и «Производитель». Многократные повторы существенно усложняют их обновление. Допустим, мы решим вместо классификатора «Память» использовать более точное название «Модуль памяти». В этом случае, чтобы внести необходимые правки, необходимо

просмотреть все строки исходной таблицы. Это пример *аномалии редактирования* данных.

Нормализация – это процесс последовательного преобразования таблиц базы данных к виду, принятому в реляционной модели данных.

Специалисты различают пять последовательных этапов (форм) приведения данных к реляционному виду от первой нормальной формы (First Normal Form, 1NF) до пятой (5NF). На каждой из ступеней нормализации таблица приобретает новые черты, которые потребуются для перехода к очередному этапу.

7.1 Первая нормальная форма

Среди требований к реляционным таблицам важнейшим принципом их построения является атомарность (одна ячейка таблицы – одно значение).

Практически любой человек, делающий первые в проектирования БД, повторяет одну и ту же ошибку – собирает всю подлежащую хранению информацию в одну таблицу.

Грубейшая ошибка проектирования – это размещение в реляционной таблице повторяющихся групп данных. Повторяющаяся группа появляется каждый раз, когда в одну ячейку таблицы пытаемся вставить больше одного значения. Например, в текстовом столбце классификатора товаров (допустим, для строки с микроволновой печью) написать следующее: «Бытовая техника; Кухонная техника; Мелкая бытовая техника; Печи микроволновые».

В рассмотренном выше примере также нарушено правило атомарности значения. В столбце «Накладная» совместно хранится два разнотипных значения: номер накладной и дата.

Под ***атомарностью значения*** понимается, с одной стороны, его целостность, а с другой – неделимость. Разделение данных на «атомы» впоследствии позволит обеспечить в БД дополнительный сервис –

разнообразные возможности сортировки данных, расширенные методы фильтрации и поиска данных.

Степень неделимости данных определяется задачами и целями разработчика, требованиями конечных пользователей и возможностями их корректной, подлежащей однозначной интерпретации записи.

Например, для ведения БД личных дел сотрудников компании данные «ФИО» необходимо разместить в трех столбцах и прописывать полностью без сокращений («Сидоров» «Иван» «Петрович»). Если же речь идет о БД кинотеатра и необходимо указать как атрибут фильма «ФИО» режиссера в большинстве случаев выделить одну ячейку.

Таблица приведена к первой нормальной форме, если в ней отсутствуют повторяющиеся группы и все значения, хранимые в ней, неделимы (атомарны).

Итак, в результате приведения рассмотренной выше таблицы учета товаров к 1NF мы получим перечень столбцов, представленный на рисунке 43.

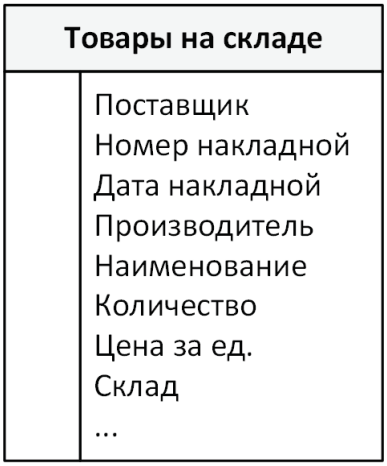


Рис. 43 – Таблица склада в состоянии 1NF

Таблица в 1NF становится громоздкой, т.к. появляется большое количество столбцов и повторяемых значений (чрезмерная избыточность данных).

Собрав нормализованную таблицу, разработчик переходит к приведению отношения к более высоким нормальным формам, для чего входящие в отношение атрибуты проверяются на функциональную зависимость между собой.

7.2 Функциональная зависимость атрибутов

Функциональные зависимости – это те связи, которые могут возникнуть между атрибутами отношений. *Один атрибут отношения А функционально зависит от другого В, если каждому значению А однозначно соответствует значение В.*

Другими словами, если в столбце А таблицы будут вноситься одинаковые значения, то в соответствующем столбце В также окажутся одинаковые данные. В символической форме функциональная зависимость записывается следующим образом: А → В. Левую часть такой записи называют *детерминантом*, а правую – *зависимой частью*.

Степень функциональной зависимости может оказаться разной:

– частичная зависимость;

– полная зависимость;

– транзитивная зависимость;

– многозначная зависимость.

Частичная зависимость возникает при наличии взаимосвязи между отдельным атрибутом и группой атрибутов отношения, выполняющих роль ключа отношения. Это тот случай, когда какой-то из атрибутов отношения окажется в функциональной зависимости только от определенной части составного ключа, а не от всего ключа в целом.

На второй ступени нормализации необходимо устранить все частичные зависимости и добиться полной функциональной зависимости (т. е. каждый столбец, не входящий в состав ключа таблицы, должен полностью от него зависеть).

Полная зависимость представляет собой прямую противоположность частичной. Примером полной функциональной зависимости может стать, с одной стороны, индивидуальный номер налогоплательщика (ИНН), а с другой – фамилия, имя и отчество обладателя ИНН. При наличии функциональной зависимости между двумя атрибутами справедливо как прямое A → B, так и обратное B → A утверждение. То есть по ИНН можно узнать ФИО человека, а по ФИО – выяснить ИНН. Полную зависимость между атрибутами обозначают следующим образом: A ↔ B.

Транзитивная зависимость возникает между двумя атрибутами, когда они связаны друг с другом через посредника. Например, когда атрибут A зависит от B, а B, в свою очередь, зависит от C: A → B → C, но обратная зависимость A ← B ← C отсутствует.

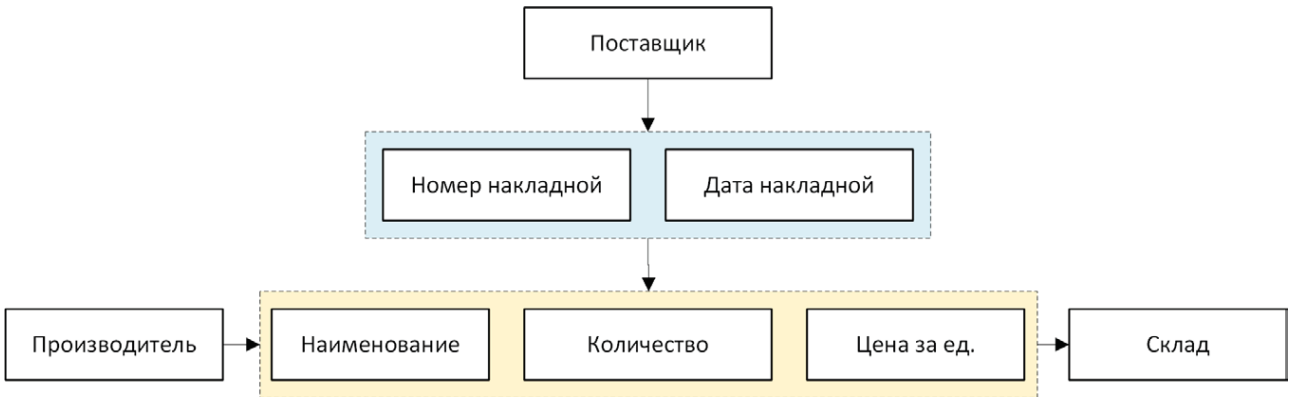
Наличие транзитивной или многозначной зависимости между атрибутами одного отношения можно считать достаточным условием, подтверждающим необходимость разнесения этих атрибутов (столбцов таблицы) по разным отношениям.

Многозначная зависимость имеет место в ситуации, когда одному значению атрибута A соответствует несколько значений атрибута B.

Наряду с зависимыми существуют и *независимые атрибуты,* когда между двумя атрибутами нет прямой связи. Например, в рассмотренном примере к независимым атрибутам можно отнести поставщика и склад. Для обозначения независимости A от B применяют запись: A ¬→ B. Соответственно, если B также не зависит от A: B ¬→ A.

Взаимную независимость двух атрибутов обозначают: A¬= B.

На рисунке 44 представлена схема зависимостей между атрибутами проекта базы данных склада. Процесс установления функциональной зависимости связан с порядком назначения ключевых полей таблицы. В рассмотренном отношении (приведенном к 1NF) пока нет ключевых атрибутов, однако с их появлением все оставшиеся неключевые атрибуты станут в той или иной степени зависеть от ключа таблицы.



.

Рис. 44 – Схема зависимостей между атрибутами

7.3 Порядок определения первичного ключа

Ключом отношения может быть набор атрибутов, который полностью и однозначно определяет значения всех остальных неключевых атрибутов. Между атрибутами, входящими в состав первичного ключа, и всеми остальными неключевыми атрибутами должна существовать полная функциональная зависимость.

Встречаются случаи, когда одна таблица содержит несколько вариантов сочетаний атрибутов, которые могут быть ключевыми (например, автомобиль можно идентифицировать номером кузова и номером двигателя), тогда говорят, что отношение содержит несколько *потенциальных ключей*.

Рассмотрим наличие потенциальных ключей в таблице учета товаров на складе. Изначально можно предположить, что потенциальным ключом является поле с номером накладной, но ведь этот же номер может повториться в будущем году. Также не достаточно для построения ключа и столбцов с номером и датой приходного ордера.

Отметим, что на рисунке 42, по накладной № 1 было получено два наименования товара (что соответствует двум строкам таблицы). Таким образом, в состав ключа придется включить три столбца: номер накладной, дата накладной, наименование.

Проблема таблиц, содержащих сложные составные ключи – частичная зависимость некоторых атрибутов таблицы от части первичного ключа. Частичная зависимость может возникнуть только в той таблице, у которой ключ составной. Поэтому разработчики баз данных в качестве первичного ключа стараются использовать только один атрибут. Чаще всего для этих целей в таблицу внедряется дополнительный столбец, не несущий никакой полезной нагрузки с точки зрения хранящейся в таблице информации, но содержащий заведомо уникальные значения. В простейшем случае это целочисленное поле автоинкрементного типа (счетчик). Первой появившейся в таблице строке присваивается 1, второй – 2 и т. д. Автоинкрементное поле гарантирует, что даже при удалении кортежа из таблицы его уникальный номер не будет назначен какой-либо из вновь вводимых строк.

7.4 Вторая нормальная форма

Таблица приведена ко второй нормальной форме, если она приведена к 1NF и в ней отсутствуют частичные зависимости. Другими словами, у такой таблицы не должно быть атрибутов, зависящих только от части первичного ключа.

В примере учета товаров на складе был определен ставной первичный ключ таблицы на основе атрибутов «Номер накладной», «Дата накладной» и «Наименование».

Наличие составного ключа привело к тому, что атрибут «Поставщик» зависит от первой половины ключа («Номер накладной» и «Дата накладной»), а атрибут «Производитель» зависит от другой части ключа – «Наименование». Следовательно, данная исходная таблица не соответствует требованиям 2NF. В таблицу необходимо добавить еще один столбец «ПКлюч», представляющий собой поле-счетчик (рис. 45).

Товары на складе	
PK	ПКлюч
	Поставщик Номер накладной Дата накладной Производитель Наименование Количество Цена за ед. Склад ...

Рис. 45 – Таблица «Склад» в состоянии 2NF

Таблица приведена ко второй нормальной форме, если она соответствует 1NF и в ней нет частичных зависимостей, т. е. нет неключевых столбцов, зависящих от части первичного ключа.

7.5 Третья нормальная форма

Несмотря на наличие первичного ключа, в рассмотренном примере (рис. 45) существуют транзитивные зависимости (связь между атрибутами через «посредника» A → B → C).

Очевидный пример транзитивности – «Поставщик». Указанный атрибут непосредственно связан лишь с атрибутами, описывающими накладную («Номер накладной» и «Дата накладной»), а все остальные атрибуты связаны с «Поставщик» транзитивно (рис. 44).

Еще один фактор, свидетельствующий о транзитивной зависимости, – возникновение связи «один ко многим» между атрибутами таблицы. В нашем случае поставщик может осуществить много поставок, соответственно, между поставщиком и накладными имеется связь 1:M.

Таблица приведена к третьей нормальной форме, если она соответствует 2NF и в ней отсутствуют транзитивные зависимости.

Транзитивная зависимость A → B → C устраняется за счет выделения атрибутов B → C (или A → B) в другую таблицу. Связи между такими таблицами строятся по ключевым столбцам: первичный ключ главной таблицы соединяется с внешним ключом подчиненной таблицы. В результате атрибут «Поставщик» выделяется в отдельную таблицу «Поставщики», то же самое происходит и с атрибутами накладной – они переносятся в таблицу «Приходные накладные» (рис. 46).

Таким образом, внутри приведенной к 3NF таблицы должны отсутствовать связи «один ко многим». В нашем случае преобразование таблицы из 2NF в 3NF приводит к декомпозиции исходного отношения в набор из 5 отношений. Здесь можно выделить 3 главные таблицы: «Поставщики», «Производители» и «Классификатор». Перечисленные таблицы называются главными потому, что они не зависят ни от каких других таблиц, т.е. они представляют сильный тип сущности). Зачастую подобные таблицы называют «справочными», так как они являются поставщиками справочных данных для подчиненных по отношению к ним таблицам.

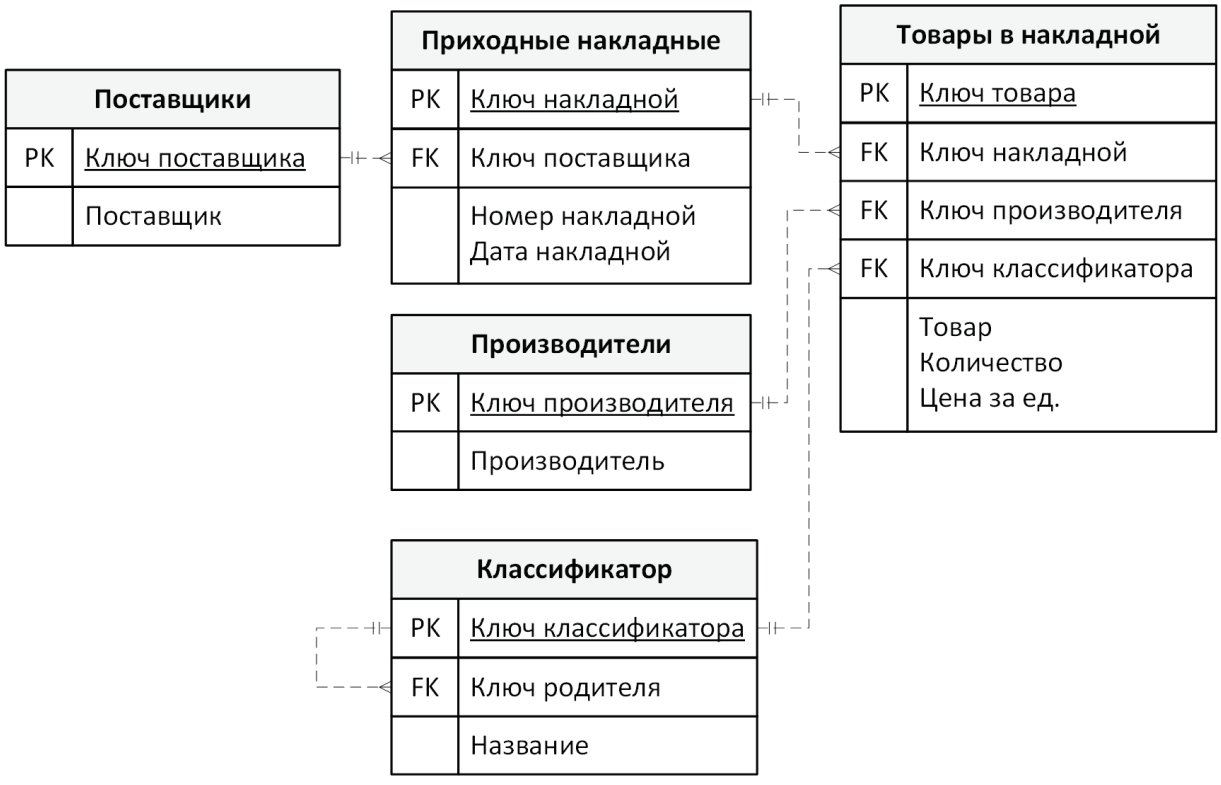


Рис. 46 – Фрагмент БД с таблицами, приведенными к 3NF

Благодаря справочным таблицам:

– существенно сокращается физический размер дочерней таблицы. Потому что теперь вместо реальных данных (названий поставщиков, названий производителей, классификации товара), в подчиненной таблице требуется хранить значение внешнего ключа;

– при редактировании значения (например, переименование поставщика) нет необходимости просматривать все записи, а достаточно изменить одно значение в справочной таблице;

– при вводе данных значительно снижается вероятность появления ошибки пользователя, так как вместо ввода с клавиатуры ему предлагаются на выбор заранее подготовленные значения.

7.6 Нормальная форма Бойса-Кодда

Помимо 3NF, специалисты выделяют усиленную разновидность 3NF – нормальную форму Бойса-Кодда 3NFBC (Boyce-Codd). Смысл усиления в том, что во время формулирования первоначальных требований к 3NF Кодд не предусмотрел вероятность того, что в нормализуемом отношении может существовать более одного потенциального ключа, указанные ключи окажутся составными и эти ключи станут обладателями хотя бы одного общего атрибута.

Вероятность совместного возникновения перечисленных событий крайне невысока, но не исключена. Поэтому и предусмотрено более строгое определение 3NFBC.

Таблица приведена к третьей нормальной форме Бойса-Кодда, когда детерминанты всех ее функциональных зависимостей являются потенциальными ключами.

При этом детерминантом функциональной зависимости является левая часть символической записи A → B.

Использовать нормальную форму Бойса-Кодда стоит только в ситуации, когда в таблицах активно применяются составные ключи. Если же при

проектировании БД разработчик опирается на идею искусственных (например, автоинкрементных), состоящих из одного атрибута ключей, в применении 3NFBC нет необходимости.

7.7 Четвертая нормальная форма

Если третья нормальная форма обеспечивает исключение транзитивных зависимостей, то 4NF нацелена на освобождение многозначных зависимостей, т.е. от связей «многие ко многим».

Итак, база данных, соответствующая четвертой ступени нормализации, обязана избавиться от многозначных зависимостей между атрибутами отношений.

Поскольку реляционная модель поддерживает только связи «один к одному» (1:1) и «один ко многим» (1:M), для реализации связи «многие ко многим» (M:N) необходимо ввести дополнительную таблицу, которая разделит связь M:N на две 1:M. Аналогичные действия проводились при построении ER-модели.

Предложенный на рисунке 46 фрагмент БД пока описывает только фрагмент БД, если точнее – накладную, по которой товар поступает на предприятие, но в логической модели не решена задача перераспределения товара внутри предприятия. Поэтому необходимо доработать БД.

Во-первых, вся поступающая по приходным накладным продукция должна быть закреплена на складе по умолчанию (основной склад предприятия).

Во-вторых, должен существовать механизм передачи товара с основного склада в различные подразделения предприятия (другие склады, магазины, цеха).

Теперь рассмотрим бизнес-логику работы БД в момент приема товаров. Допустим, по приходной накладной на основной склад предприятия поступает 10 холодильников, об этом создается соответствующая запись в таблице «Товары в накладной». Вполне вероятно, что пара холодильников из этой

партии будет немедленно отправлена в магазин, а остальные 8 переместятся на другой склад (склады) нашей фирмы. Таким образом, между таблицей «Товары в накладной» и таблицей «Склады» (ее пока еще нет на рис. 46) формируется связь M:N. Решение этой задачи обеспечено путем появления между таблицами «Товары в накладной» и «Склады» коммуникационной таблицы «Перемещаемый товар», благодаря чему удалось разделить связь M:N на две связи 1:M (Рис. 47).

Рис. 47 – Улучшенная БД с таблицами, приведенными к 4NF

Таблица приведена к четвертой нормальной форме, если она соответствует 3NF и в ней отсутствуют многозначные зависимости (многие ко многим).

7.8 Пятая нормальная форма

Для обычного разработчика БД пятая нормальная форма представляет скорее теоретический, нежели практический интерес. 5NF требует обеспечения беспрепятственной возможности перестройки данных в нормализованных

таблицах. Приведение таблицы к высшей степени нормализации – крайне редкий случай. Это действие имеет смысл, только если таблица содержит так называемые зависимые сочетания.

Зависимые сочетания – это свойство декомпозиции, которое вызывает генерацию ложных строк при обратном соединении декомпозированных отношений с помощью операции естественного соединения.

Предположим, что в базе данных существует некое отношение, хранящее данные о поставщике, товаре (полученном от этого поставщика) и производителе, изготовившем товар (рис. 48).

Вполне вероятно, что в процессе работы с базой данных на основе исходной таблицы появится необходимость построить два запроса:

1) первый запрос возвращает данные о поставщиках и товарах и сохранен в виде таблицы TABLE2);

2) на базе второго запроса получена информация о товарах и производителях (таблица TABLE3).

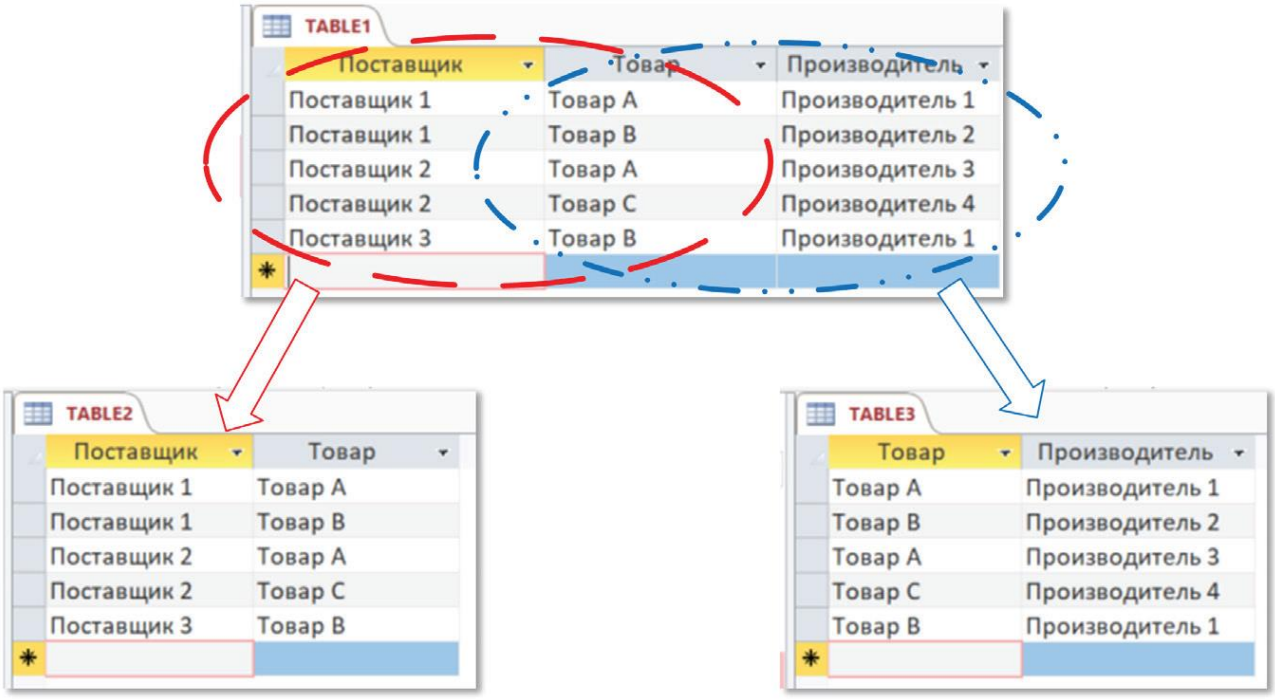


Рис. 48 – Декомпозиция таблицы

Обратим внимание, что данные в новых таблицах вполне корректны и соответствуют значениям, содержащимся в исходной таблице.

А теперь попробуем решить обратную задачу – восстановим исходную таблицу TABLE1 на основе данных из TABLE2 и TABLE3. Для этого создаем запрос на языке SQL, объединяющий обе таблицы по общему для них аргументу «Товар»:

```
SELECT TABLE2.Поставщик, TABLE2.Товар, TABLE3.Производитель
FROM TABLE2 INNER JOIN TABLE3 ON
TABLE3.Товар=TABLE2.Товар;
```

После соединения двух таблиц получены четыре ложные (не существующие в реальной таблице) строки (рис. 49).

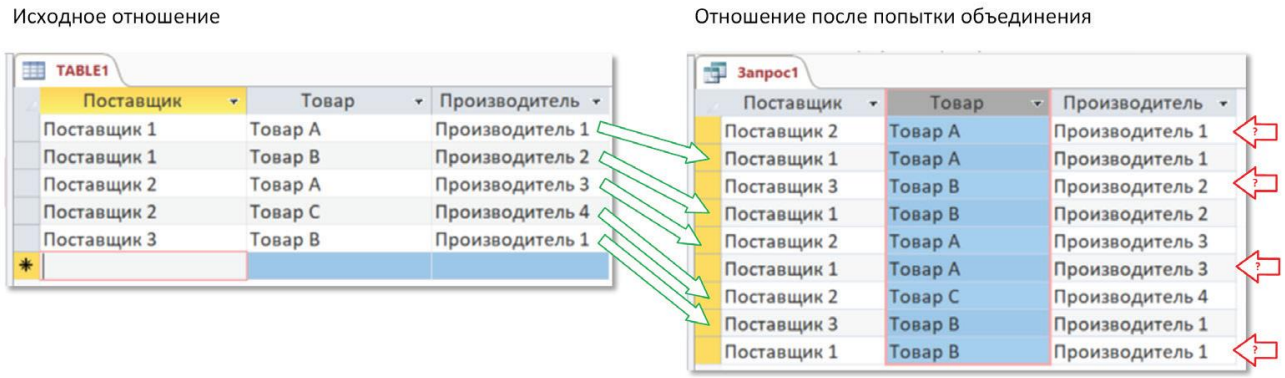


Рис. 49 – Генерация ложных строк после объединения

7.9 Доменно-ключевая нормальная форма

Нормальные формы вводились исследователями начиная с 1970-х годов с целью устранить выявленные ими аномалии данных. И ввод каждой очередной НФ устранял определенную группу аномалий, но буквально сразу находилась новая подлежащая устранению аномалия.

Однако этот процесс пришел к логическому финалу в 1981 году, когда Р. Фагин (R. Fagin) в своей статье ввел понятие ***доменно-ключевой нормальной***

формы (domain/key normal form, DK/NF) и доказал, что любое отношение, приведенное к DK/NF, окажется свободным от всех возможных аномалий модификации. Таким образом Фагин указал на то, что введение нормальных форм более высокого порядка (с точки зрения борьбы с аномалиями модификации) не имеет смысла.

Отношение находится в доменно-ключевой нормальной форме, если каждое ограничение, накладываемое на это отношение, является логичоеским следствием определения доменов и ключей.

Отношение находится в DK/NF, если выполнение ограничений на домены и ключи приводит к выполнению всех ограничений. Таким образом, в состояние DK/NF отношение может перейти на любой ступени нормализации, например после 2NF. И в дальнейшей нормализации больше не нуждаться.

Пока не известен ни один алгоритм преобразования отношения к доменно-ключевой нормальной форме, так что создание отношений в DK/NF является более искусством, чем наукой.

Таким образом, нормализация осуществляется на этапе логического проектирования БД и представляет собой вариант восходящего подхода, который начинается с установления связей между атрибутами. На практике для построения приемлемой логической модели БД следует пройти 4 ступени нормальных форм:

– 1NF – все поля в таблицах неделимы и не содержат повторяющихся групп;

– 2NF – все неключевые поля в таблицах зависят от первичного ключа;

– 3NF – в таблицах отсутствуют избыточные неключевые поля;

– 4NF – в таблицах устранены многозначные зависимости.

Пятая нормальная форма предназначена для устранения зависимых сочетаний и учитывается только при декомпозиции отношений.

Контрольные вопросы:

1. С какими аномалиями мы можем столкнуться при модификации данных в ненормализованной таблице?

2. На каком этапе проектирования БД осуществляется нормализация?

3. Дайте понятие термину «нормализация».

4. Какие разновидности функциональных зависимостей вам известны?

5. Дайте определение первой и второй нормальных форм. Какие аномалии они исключают?

6. Что такое первичный ключ, и какова его роль в процессе нормализации?

7. Дайте характеристику третьей нормальной формы и нормальной формы Бойса-Кодда.

8. Поясните назначение четвертой нормальной формы.

9. Что представляет собой пятая нормальная форма, и надо ли учитывать ее на этапе логического проектирования БД?

10. Почему процесс нормализации считается восходящим подходом?

ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ ПОСТРОЕНИЯ И РАБОТЫ С БАЗОЙ ДАННЫХ

1 Основы работы в MS Access. Построение и связывание таблиц

MS Access представляет собой систему обслуживания реляционных баз данных с графической оболочкой. Данные в таких базах представляются в виде одной или нескольких таблиц, состоящих из однотипных записей. Система обслуживания включает в себя ввод данных в ЭВМ, отбор данных по каким-либо признакам (критериям или параметрам), преобразование структуры данных, вывод данных, являющихся результатом решения задач в табличном или каком-либо ином удобном для пользователя виде.

MS Access позволяет создавать связанные объекты и устанавливать ссылочную целостность данных. MS Access поддерживает встраивание OLE-объектов (Object Linking and Embedding) в рамках среды Windows.

В состав пакета MS Access входит также ряд специализированных программ, решающих отдельные задачи (так называемых Мастеров).

Запуск программы

Запуск MS Access можно осуществить следующими способами:

1) меню **Пуск** системы Windows → **Все программы → Microsoft Office → Microsoft Access 2010**;

2) запустить файл с расширением *.accdb, *.accdw, *.accde, *.accdt, *.accdr, *.mdw.

После запуска MS Access на экране появляется окно диалога Access с наименованием MICROSOFT ACCESS в строке заголовка. В этом окне следует выбрать одно из предлагаемых действий:

1) открыть существующую базу данных;

2) создать новую (пустую) базу данных;

3) создать базу данных с помощью прилагаемых **Шаблонов.**

Для создания пустой базы данных выберите в диалоговом окне **Доступные шаблоны** выберите значение **Новая база данных** и в открывшемся окне диалога Новая база данных:

1) в строке **Имя файла** задайте имя новой базы данных.

2) в конце строки **Имя файла** нажмите знак Папка и задайте папку, в которой предполагается сохранить создаваемую базу данных.

3) нажмите кнопку **Создать.**

Интерфейс MS Access 2010

Главный элемент пользовательского интерфейса MS Access 2010 представляет собой *Ленту*, которая идет вдоль верхней части окна каждого приложения (рис. 50). Лента управления содержит вкладки. По умолчанию их пять: **Файл, Главная, Создание, Внешние данные, Работа с базами данных.** Каждая вкладка связана с видом выполняемого действия.

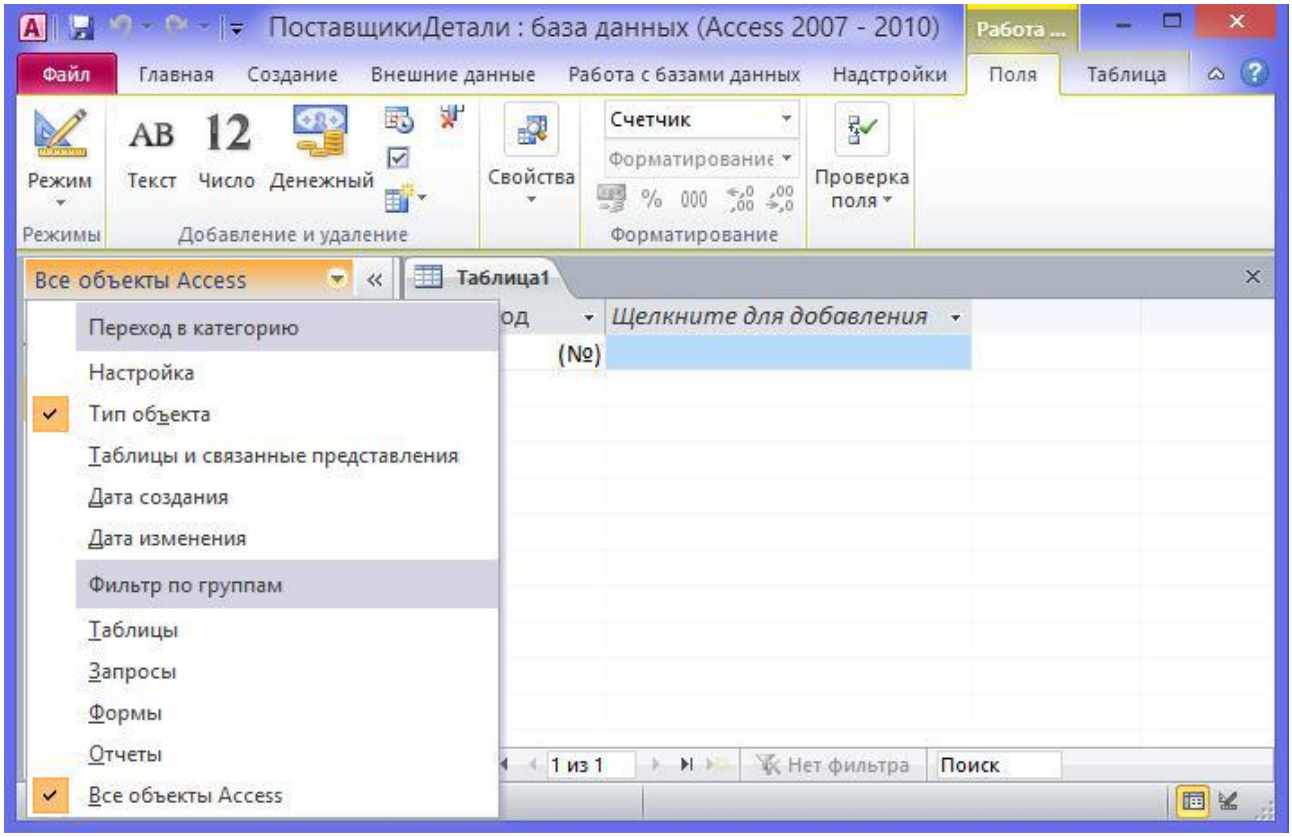


Рис. 50 – Окно базы данных MS Access 2010

Панель быстрого доступа. Расположена в верхней части окна Access. По умолчанию на панели быстрого доступа расположены четыре кнопки управления.

Область навигации, расположенная по левому краю окна Access. Она предназначена для отображения объектов или групп объектов открытой базы данных, а также для перехода от объекта к объекту. Чтобы раскрыть группу объектов следует щелкнуть мышкой по кнопке. Управлять объектами можно командами ленты и командами контекстного меню.

Область документов, в которой отображается каждый объект базы данных, открываемый в любом режиме.

Строка состояния, расположенная вдоль нижней границы окна Access, отображающая кнопки переключения в различные режимы работы с активным объектом.

Объекты базы данных Access

Таблицы – это основа БД, именно в них хранится информация. Таблицы состоят из строк и столбцов. Каждая строка таблицы – это запись, каждый столбец – поле. Таким образом, запись состоит из полей. Все элементы какого-либо столбца имеют один и тот же тип, например, числовой, текстовый и т.п. Тип поля задаётся в процессе создания структуры таблицы.

Запрос обеспечивает выборку данных, удовлетворяющих некоторым условиям, из одной или нескольких таблиц или ранее созданных других запросов.

Форма – это окно, содержащее элементы управления – такие как надписи, текстовые поля, флажки и списки для просмотра, ввода и редактирования информации. Формы могут отображать одну или несколько записей из одной или нескольких таблиц или запросов.

Отчёты предназначены для вывода на печать информации из БД в обработанном виде. Отчёт может содержать элементы управления и данные из одной или нескольких таблиц.

Страница доступа к данным напоминает форму, но хранится в отдельном файле формата HTML, поэтому её можно просматривать в Web-браузере, например, Internet Explorer и, следовательно, иметь доступ к данным из Интернета или корпоративной сети. (HTML – Hypertext Markup Language – язык разметки гипертекстов – используется для создания Web-страниц.)

Макрос представляет собой набор нескольких макрокоманд, имеющий имя, использование которого позволяет ускорить выполнение часто встречающихся действий.

Модули представляют собой программы, написанные на встроенном языке Visual Basic for Application (VBA). Они разрабатываются для выполнения действий, которые трудно осуществить с помощью визуальных инструментов или для автоматизации обработки данных, хранящимися в БД.

Для реализации сквозного примера построения БД и работы с ее ключевыми элементами, рассмотрим базу данных **Продажи**, имеющую следующую структуру данных (рис. 51).

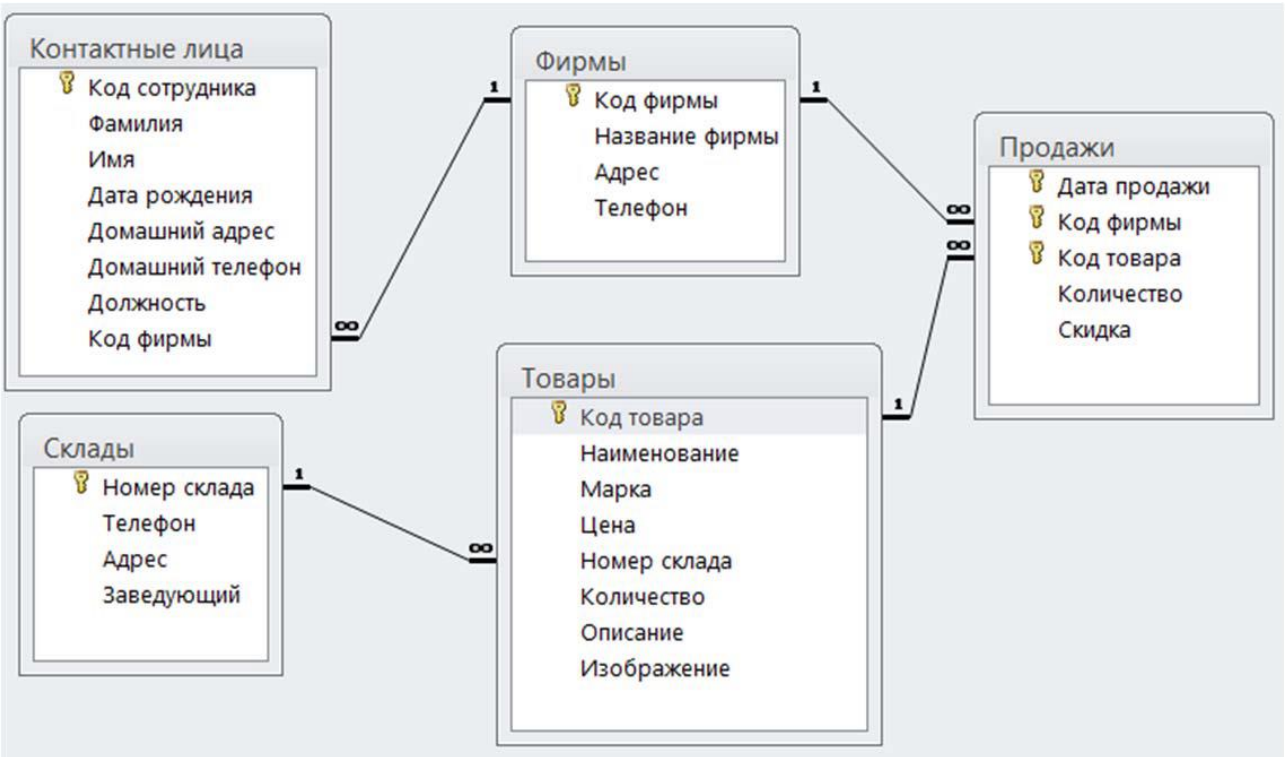


Рис. 51 – Схема БД «Продажи»

Создание таблиц

Создание таблицы производится в два этапа:

– определение структуры таблицы;

– ввод данных.

При создании новой базы данных MS Access 2010 автоматически входит в режим создания таблицы. Ей присваивается имя **Таблица 1.**

При необходимости добавить новую таблицу в базу данных: вкладка **Создание** – группа **Таблицы** – кнопка **Таблицы**.

Структура таблицы может быть создана с использованием **режима Таблицы** либо в **режиме Конструктора**.

Наиболее широкие возможности по определению параметров создаваемой таблицы предоставляет режим Конструктора (рис. 52).

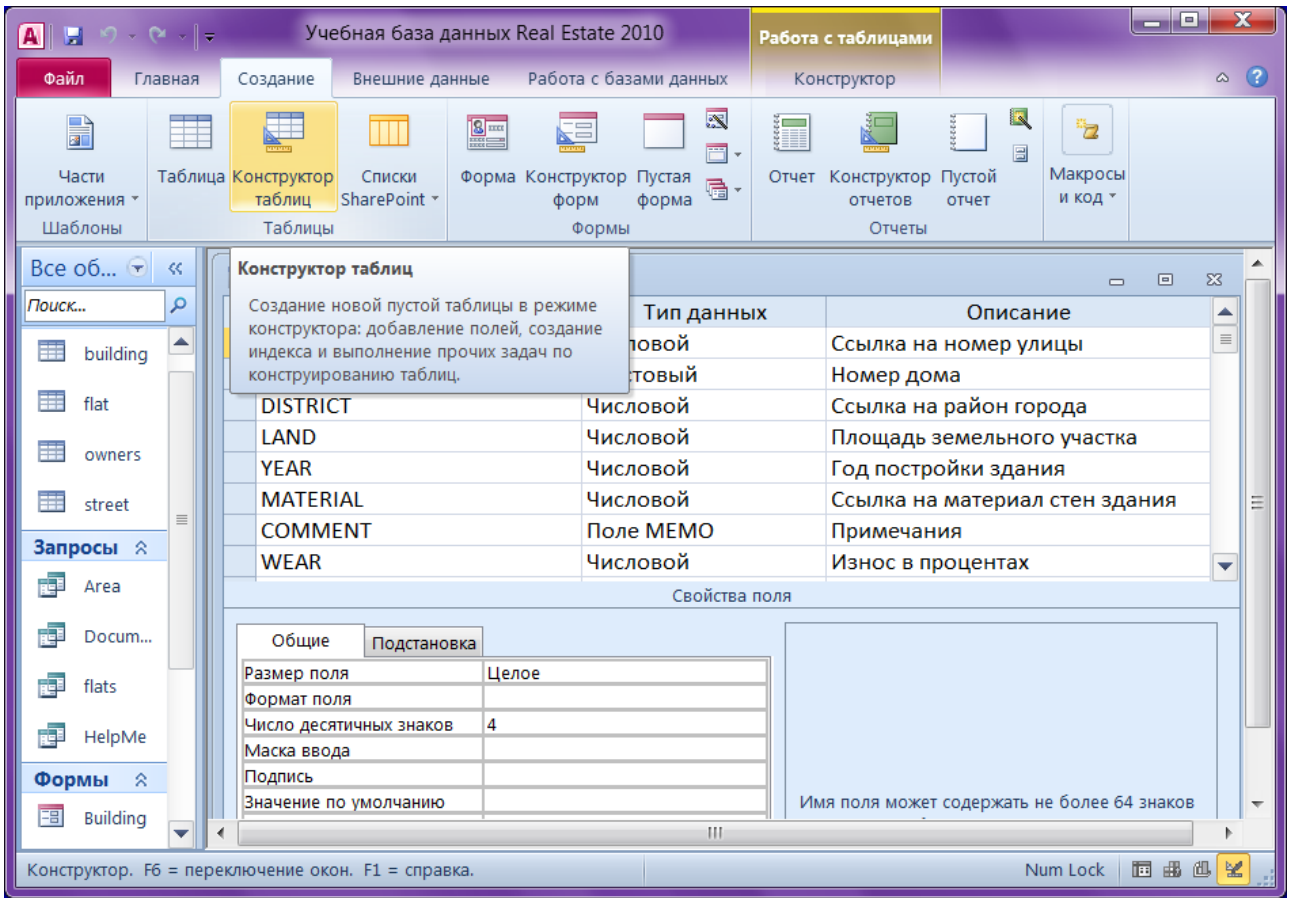


Рис. 52 – Пример построения таблицы в режиме конструктора

При определении структуры таблицы устанавливается, из каких полей состоит отдельная запись БД, и задается тип данных каждого поля (табл. 2).

Таблица 2 – Типы данных полей таблиц MS Access 2010

Тип данных	Описание
Текстовый	Текст или числа, не требующие проведения расчетов. Максимальная длина – 255 символов. По умолчанию длина текстового поля устанавливается равной максимальной длине
Поле MEMO	Предназначены для хранения больших текстовых данных. Длина поля может достигать 64 Кбайт. Поле не может быть ключевым или индексированным. Поля MEMO полезны для хранения больших объемов информации. При работе с Office Access 2010 можно задать свойство **Только добавление**, при котором приложение Access сохраняет историю всех изменений поля MEMO. Историю изменений затем можно просмотреть
Числовой	Содержит множество подтипов (размеров). От выбора размера зависит точность вычислений. Используйте целый тип для полей, которые используются в ссылках на другие таблицы базы данных
Счетчик	Уникальные, последовательно возрастающие числа, автоматически вводящиеся в таблицу при добавлении каждой новой записи
Логический	Содержит одно из двух возможных значений 0 – для представления значения «нет» и –1 (обратите внимание: минус 1) для «да»
Денежный	Позволяет выполнять расчеты с точностью до 15 знаков в целой и до 4 знаков в дробной части
Дата/Время	Семь видов форматов для отображения даты и времени
Гиперсвязь	Содержит адреса Web-страниц Используется для хранения гиперссылок вызова веб-страниц одним щелчком с помощью URL-адреса или файлов с помощью формата универсального имени UNC. Кроме того, можно использовать ссылку на объекты Access, хранящиеся в базе данных
Поле объекта OLE	Включает рисунок, фотографию, звукозапись, диаграммы, векторную графику, форматированный текст и т. п.
Вложение	Позволяет хранить документы и двоичные файлы любых типов в базе данных без излишнего увеличения ее объема. Для уменьшения общего объема данных вложения автоматически сжимаются. Этот тип данных используется, например, если нужно вложить в запись документ Microsoft Office Word 2007 или сохранить в базе данных набор цифровых изображений. В одной записи можно хранить несколько вложений
Мастер подстановок	Фактически типом данных не является, используется для запуска мастера подстановок, с помощью которого можно создать поле, позволяющее выбрать значение из другой таблицы, запроса или списка значений, используя поле со списком.

Каждое поле идентифицируется своим именем. Кроме этих атрибутов, каждое поле таблицы обладает дополнительными свойствами, отображаемыми в нижней части конструктора и определяющими условия ввода данных.

Имена полей печатаются в клетках столбца **Имя** поля конструктора таблиц. Имя поля может содержать до 64 символов, включая пробелы, за исключением точки, восклицательного знака и квадратных скобок.

В столбце **Тип данных** определяется тип данных в этом поле (Переход между столбцами осуществляется с помощью клавиши **Tab**). По умолчанию Microsoft Access присваивает полю текстовый тип данных. Щелкнув на стрелку в правой части клетки, можно выбрать нужный тип данных из открывшегося списка.

В столбце **Описание** печатаются комментарии, описывающие данное поле. Описание поля используется при обращении к полю в дальнейшем. При вводе данных в это поле текст описания выводится в строку состояния.

Установка первичного ключа

Объявление первичного ключа обеспечивает уникальность строк и препятствует вводу повторяющихся блоков данных. Это поле не может содержать одинаковую величину в двух различных записях. Ключевое поле помогает Microsoft Access наиболее активно организовать поиск, хранение и объединение данных.

В Microsoft Access можно выделить три типа ключевых полей: *счетчик, простой ключ и составной ключ*.

Указание *поля счетчика* в качестве ключевого является наиболее простым способом создания ключевых полей. Если до сохранения созданной таблицы ключевые поля не были определены, то при сохранении будет выдано сообщение о создании ключевого поля. При нажатии кнопки **Да** будет создано ключевое поле счетчика.

Простой ключ определяется полем, содержащим уникальные значения, такие как коды или инвентарные номера. Ключевое поле не может содержать

повторяющиеся или пустые значения. Если устранить повторы путем изменения значений невозможно, то следует либо добавить в таблицу поле счетчика и сделать его ключевым, либо определить составной ключ.

В случаях, когда невозможно гарантировать уникальность значений каждого поля, существует возможность создать *составной ключ*, состоящий из нескольких полей. Чаще всего такая ситуация возникает для таблицы, используемой для связывания двух таблиц в отношении «многие-ко-многим».

Если определить подходящий набор полей для составного ключа сложно, следует добавить поле счетчика и сделать его ключевым. Например, не рекомендуется определять ключ по полям «Имена» и «Фамилии», поскольку нельзя исключить повторения этой пары значений для разных людей. Обычно в качестве ключа используются числовые поля.

Первичный ключ может быть определен только в режиме конструктора таблиц путем реализации следующих шагов:

1) выделите поле, которое должно стать полем первичного ключа;

2) вкладка **Конструктор** – группа **Сервис** – кнопка **Ключевое поле.**

Установка характеристик поля. В нижней части окна Конструктора таблиц указываются свойства каждого поля таблицы (табл. 3). Каждый тип данных связан с вполне определенным набором свойств. Данные всех типов имеют свойство **Подпись поля**. Это свойство позволяет дать столбцу название, отличное от названия соответствующего поля.

Таблица 3 – Свойства полей таблицы MS Access 2010

Свойство	Назначение
1	2
Размер поля	Задает максимальное число символов для ввода в данное поле
Новые значения	Определяет способ изменения значений счетчика при добавлении новых записей
Формат поля	Задает формат вывода значений данного поля
Число десятичных знаков	Определяет число десятичных знаков, используемых при отображении чисел
Маска ввода	Задает маску ввода, облегчающую ввод данных в поле

Продолжение табл. 3

1	2
Подпись	Определяет текст, который выводится в качестве подписи поля
Значение по умолчанию	Позволяет указать значение, автоматически вводящееся в поле при создании новой записи
Условие на значение	Выражение, накладывающее ограничение на значения, которые вводятся в данное поле
Сообщение об ошибке	Позволяет указать текст сообщения, выводящегося на экран, если введенные данные нарушают условие, определенное в свойстве Условие на значение
Обязательное поле	Указывает требует ли поле обязательного ввода значения
Пустые строки	Определяет допускается ли ввод в данное поле пустых строк
Индексированное поле	Определяет индекс, создаваемый по одному полю

Индекс – средство Microsoft Access, ускоряющее поиск и сортировку в таблице. Ключевое поле таблицы индексируется автоматически. Не допускается создание индексов для полей типа MEMO, Гиперссылка и объект OLE.

Для задания структуры маски ввода используется специальный набор символов (табл. 4).

Таблица 4 – Символы маски ввода

Символ	Описание
0	Цифра (0-9, обязательный знак; знаки (+) и (-) не разрешены)
9	Цифра или пробел (необязательный знак; знаки (+) и (-) не разрешены)
#	Цифра или пробел (необязательный знак; незаполненные позиции выводятся как пробелы в режиме редактирования, но удаляются при сохранении данных; знаки (+) и (-) не разрешены)
L	Буква (от А до Я, обязательный знак)
?	Буква (от А до Я, необязательный знак)
A	Буква или цифра (обязательный знак)
a	Буква или цифра (необязательный знак)
&	Любой знак или пробел (обязательный знак)
C	Любой знак или пробел (необязательный знак)
. , : ; - /	Десятичный разделитель, разделители групп разрядов, времени или даты
<	Преобразует все знаки к нижнему регистру
>	Преобразует все знаки к верхнему регистру
!	Указывает заполнение маски ввода справа налево, а не слева направо. Восклицательный знак в маске ввода можно помещать в любую позицию
\	Указывает, что следующий знак будет отображаться как текстовая константа

Добавление, удаление и перемещение полей

Для добавления нового поля между уже существующими полями необходимо выполнить действия:

1) установите курсор в поле, перед которым хотите добавить новое поле;

2) вкладка **Конструктор** – группа **Сервис** – кнопка **Вставить строки.**

Для удаления поля из БД необходимо выполнить действия:

1) выделите всю строку поля, щелкнув курсором на серой кнопке слева от имени поля (курсор примет вид стрелки, направленной вправо):

2) вкладка **Конструктор** – группа **Сервис** – кнопка **Удалить строки.**

Для изменения порядка следования полей необходимо выполнить:

1) выделите всю строку поля, щелкнув курсором на серой кнопке слева от имени поля (курсор примет вид стрелки, направленной вправо);

2) переместите с помощью мыши строку в новое место (над тем полем, перед которым хотите расположить).

Сохранение структуры таблицы. Если структура была создана или изменена, ее необходимо сохранить, для чего существуют следующие способы:

– нажмите кнопку **Сохранить** на панели быстрого доступа;

– вкладка **Файл** – кнопка **Сохранить;**

– сочетание клавиш **Ctrl+S**.

Если таблица еще не сохранялась, то в появившемся диалоговом окне введите имя таблицы в соответствующее поле.

Если новая таблица не имеет ключевого поля, для автоматического создания ключа нажмите кнопку **ДА.**

Установление связей между таблицами

После создания таблиц их необходимо связать между собой, что реализуется путем выполнения следующих действий:

1) выберите вкладку **Работа с базами данных;**

2) в группе **Отношения** выберите **Схема данны;**

3) в появившемся диалоговом окне **Добавление таблицы** выберите таблицы, которые должны быть связаны. Названия каждой из таблиц со списками полей появятся в окне **Схема данных**;

4) установите курсор в любую из таблиц на поле, по которому будет установлена связь и «перетащите» это поле на связующее поле другой таблицы. Тип данных, значения и свойства связываемых полей должны совпадать;

5) активизируйте флажок **Обеспечение целостности данных**;

Если установить флажок **Каскадное обновление связанных полей**, то при изменении ключевого поля главной таблицы автоматически будут изменяться и соответствующие значения связанных записей.

Если установить флажок **Каскадное удаление связанных полей**, то при удалении записи в главной таблице будут удалены и все связанные записи в подчиненной таблице.

От полей, указанных при определении связи зависит тип создаваемой связи, который отображается в этом же окне.

Отношение «один-к-одному» создается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы.

Отношение «один-ко-многим» создается в том случае, когда только одно из полей является ключевым или имеет уникальный индекс. В отношении «один-ко-многим» главной таблицей является таблица, которая содержит первичный ключ и составляет часть «один» в этом отношении. Таблица со стороны «много» является подчиненной таблицей. Связующее поле (или поля) в ней с таким же типом информации как в первичном ключе главной таблицы является полем внешнего ключа.

Связь с отношением «многие-ко-многим» фактически представляет две связи с отношением "один-ко-многим" через третью таблицу, ключ которой состоит, по крайней мере, из двух полей, которые являются полями первичного ключа в двух других таблицах.

В случае если для какой-то из таблиц не было определено ключевое поле, то в поле **Тип отношения** отображается текст: «Не определено».

Для удаления связи в окне **Схема данных** выделите ненужную связь и нажмите клавишу **Delete**.

Завершение работы MS Access. Для завершения работы MS Access возможны следующие действия:

– вкладa **Файл** – кнопка **Выход**;

– кнопка **Закрыть** в верхней части окна MS Access;

– нажать сочетание клавиш **Alt+F4**.

Задание для самостоятельного выполнения

1. Создайте базу данных **Продажи**, в соответствии со структурой данных, представленной на рисунке 51.

2. Создайте таблицы «Склады», «Товары», «Фирмы», «Контактные лица» и «Продажи», определив в режиме **Конструктора** их структуры, задав ключевые поля и указав тип данных для каждого поля (табл. 5)

Таблица 5 – Структура данных БД «Продажи»

Название поля	Тип данных	Свойства поля	Ключ
1	2	3	4
Таблица Контактные лица			
Код сотрудника	Числовой	Размер: Длинное целое	+
Фамилия	Короткий текст	Размер:255	
Имя	Короткий текст	Размер: 255	
Дата рождения	Дата и время	Размер: 255	
Домашний адрес	Короткий текст	Размер: 255	
Домашний телефон	Короткий текст	Размер: 255	
Должность	Короткий текст	Размер: 255	
Код фирмы	Числовой	Размер: Длинное целое	
Таблица Продажи			
Дата продажи	Дата и время	Размер: Длинное целое	+
Код фирмы	Числовой	Размер: Длинное целое	+
Код товара	Числовой	Размер: Длинное целое	+
Количество	Числовой	Размер: Длинное целое	
Скидка	Числовой	Размер: Длинное целое	
Таблица Склады			
Номер склада	Числовой	Размер: Длинное целое	+
Телефон	Короткий текст	Размер: 50	

Продолжение табл. 5

1	2	3	4
Адрес	Короткий текст	Размер: 255	
Заведующий	Короткий текст	Размер: 255	
Таблица Товары			
Код товара	Счетчик	Размер: Длинное целое	+
Наименование	Короткий текст	Размер: 50	
Марка	Короткий текст	Размер: 255	
Цена	Денежный		
Номер склада	Числовой	Размер: Длинное целое	
Изображение	Поле объекта OLE		
Таблица Фирмы			
Код фирмы	Числовой	Размер: Длинное целое	
Название фирмы	Короткий текст	Размер: 255	
Адрес	Короткий текст	Размер: 255	
Телефон	Короткий текст	Размер: 255	

3. Сохраните таблицы под соответствующими названиями.

4. Установите связи между таблицами, подтвердив необходимость обеспечения целостности данных, каскадного обновления и удаления данных в связанных таблицах.

2 Работа с данными таблицы

Переход на нужное поле или запись

Для перехода между столбцами и к следующей записи используется клавиша **Tab** или комбинация клавиш **Shift+Tab**.

Для перехода между записями также служат кнопки переходов в нижнем левом углу окна, где также отображается общее количество записей и номер текущей записи.

Для перехода к конкретной записи вместо номера текущей записи нужно ввести требуемый номер и нажать клавишу **Enter**.

Переход к другой записи также может быть осуществлен с помощью команды **Перейти** (вкладка **Главная** – группа **Найти**).

Быстрый путь ввода данных

Для копирования данных из аналогичного поля предыдущей записи в текущую надо нажать **<Ctrl>+"** (кавычки).

Вставка текущего времени или даты.

Чтобы вставить текущую дату надо нажать **<Ctrl>+Ж.**

Чтобы вставить текущее время надо нажать **<Ctrl>+<Shift>+Ж.**

Для экономии времени при вводе данных также можно пользоваться инструментами редактирования Windows: вырезанием **<Ctrl>+<X>,** копированием **<Ctrl>+<C>** и вставкой **<Ctrl>+<V>** в буфер.

Сохранение данных

В MS Access изменения сохраняются автоматически при следующих действиях:

– переход к следующей записи;

– закрытие режима таблицы или формы.

Чтобы сохранить запись без перехода к другой записи выполните команду **Сохранить** (**Ctrl+S**).

Добавление и удаление записей

Обычно таблица имеет в конце пустую запись для добавления новых данных.

Для добавления данных в новую запись:

1) перейдите на первое пустое поле новой записи;

2) введите значение и нажмите клавишу **Tab** для перехода к следующему полю.

Для удаления записей:

1) выделите записи для удаления, щелкнув курсором на серой кнопке слева от первой удаляемой записи и переместив указатель вдоль требуемых записей;

2) нажмите клавишу **Delete** или выберите вкладка **Главная** – группа **Записи** – кнопка **Удалить**.

Вставка в запись рисунка или объекта

Рисунок или объект добавляется из имеющегося файла либо создается в приложении OLE (например, в MS Paint), а затем вставляется в текущую запись.

Чтобы добавить рисунок или любой другой объект в запись:

1) перейдите в режим конструктора таблиц;

2) добавьте поле объекта OLE;

3) в режиме таблицы установите курсор в нужную клетку, правой клавишей вызовите контекстное меню и выполните команду **Вставит объект**.

Если объект вставляется из существующего файла:

1) в появившемся окне выберите переключатель **Создать из файла**;

2) введите полное имя добавляемого файла в поле **Файл** или нажмите кнопку Обзор и выберите имя требуемого файла;

3) нажмите кнопку **ОК**.

Если объект нужно создать, то:

1) выберите тип создаваемого объекта в поле **Тип объекта** (например, Bitmap Image);

2) нажмите кнопку **ОК**;

3) после создания рисунка или объекта в приложении OLE выполните команду **Выход** из приложения OLE.

Просмотр данных в виде формы

Просмотр БД в виде формы позволяет видеть только одну запись. Для автоматического создания простой формы необходимо: вкладка **Создание** – группа **Формы** – кнопка **Форма**.

Добавление записей с помощью формы

Добавлять записи в БД удобно, используя формы ввода. Для этого в окне формы следует щелкнуть на кнопке «Новая запись» и ввести новые данные в пустые поля формы.

Поиск и замена данных

Поиск информации в БД реализуется согласно следующему алгоритму действий:

1) в окне **Все объекты** Access выберите таблицу или форму, а затем дважды щелкните на имени соответствующего объекта, в котором хотите осуществить поиск;

2) щелкните в любом месте поля, в котором будет осуществляться поиск;

3) щелкните на кнопке **Найти** (вкладка **Главная** – группа **Найти**). Появляется диалоговое окно **Поиск и замена**;

4) в поле **Образец** введите последовательность символов, которую нужно искать;

5) в поле **Совпадение** укажите:

– **С начала поля**, если данные по которым ведется поиск известны целиком;

– **С любой частью поля**, если поиск ведется по части данных, которая может оказаться в различных областях поля, например, по первым или последним трем символам;

6) установите область и направление поиска;

7) щелкните на кнопке **Найти далее**;

8) если осуществляется поиск более чем одной записи, то для продолжения поиска снова щелкните на кнопке **Найти далее**;

9) если других записей не найдено, Microsoft Access запрашивает, намерены ли Вы продолжить поиск, начав с самой первой записи таблицы;

10) щелкните на кнопке **Закрыть**, чтобы закрыть диалоговое окно.

Аналогично для выполнения замены данных используйте вкладку **Замена** в диалоговом окне **Поиск и замена**.

Сортировка данных

Для выполнения сортировки данных в таблице или форме необходимо выполнить действия:

1) выберите в таблице или форме поле сортировки. В режиме таблицы выделите столбец для сортировки;

2) для выполнения сортировки по возрастанию (А-Я) или по убыванию (Я-А) нажмите соответствующую кнопку из группы **Сортировка и фильтр** вкладка **Главная**.

Фильтрация данных

Фильтрация – удобный способ отображения нужных данных. Фильтры позволяют просмотреть только отдельные записи в форме, отчете, запросе или таблице либо напечатать некоторые записи из отчета, таблицы или запроса. С помощью фильтра можно ограничить объем отображаемых данных, не изменяя макет базовых объектов. Так как после применения фильтра представление содержит только записи с выбранными значениями, остальные записи скрываются до очистки фильтра. Для столбцов таблиц и элементов управления в формах и отчетах, связанных с выражениями, фильтрация не поддерживается.

Существует несколько типов фильтров.

Обычные фильтры встроены в каждое представление Access 2010. Доступность команд фильтра зависит от типа и значений поля. Для каждого типа данных предусмотрено несколько готовых фильтров. Они доступны в виде команд меню в режимах таблицы и макета и в представлениях формы и отчета. Таблицу или форму можно отфильтровать не только с помощью этих фильтров, но и путем заполнения формы (фильтр по форме).

Пользователь, который может написать выражение самостоятельно, может добиться большей гибкости, создав собственные фильтры с помощью параметров вкладки документа **Фильтр**.

Обычные фильтры используются для фильтрации по значению или диапазону значений.

Фильтрация по выделенному позволяет отсортировать все строки в таблице, содержащие значение, которое совпадает с выделенным значением в строке. Используется в режиме таблицы.

Фильтр по форме используется, если требуется отфильтровать несколько полей в форме или таблице либо найти конкретную запись.

Расширенный фильтр позволяет задать пользовательские условия фильтра.

Алгоритм применения обычного фильтра:

1) откройте таблицу, запрос, форму или отчет в режиме таблицы, формы, отчета или макета;

2) убедитесь, что представление еще не отфильтровано. В области маркировки проверьте наличие значка **Без фильтра** или затененного значка **Нет фильтра**. Чтобы удалить все фильтры для определенного объекта, на вкладке **Главная** в группе **Сортировка и фильтр** нажмите кнопку **Дополнительно** и выберите команду **Очистить все фильтры**;

3) щелкните в любом месте столбца или элемента управления, соответствующего первому полю, к которому требуется применить фильтр, и на вкладке **Главная** в группе **Сортировка и фильтр** нажмите кнопку **Фильтр**.

Чтобы применить обычный фильтр, выберите пункт **Текстовые (или Числовые, Даты) фильтры** и выберите нужный фильтр. Для фильтров **Равно** и **Между** потребуется ввести нужные значения.

Некоторые символы, например *, % и ?, в текстовом поле фильтра считаются специальными знаками. Так, звездочка (*) представляет строку знаков, поэтому строка «а*» соответствует любой строке, начинающейся с буквы «а», а не только строке «а*». Если не требуется, чтобы знак считался

специальным, необходимо заключить его в квадратные скобки «[]», например «a[*]». В базах данных, использующих стандарт ANSI-89, к специальным относятся знаки: *, ?, [,], !, -, #. В базах данных стандарта ANSI-92 специальными считаются знаки: %, _, [,], ^, -. В Access можно использовать любой из этих стандартов, но не оба стандарта одновременно.

Чтобы применить фильтр на основе значений поля, снимите флажки возле значений, для которых не следует применять фильтр, и затем нажмите кнопку **ОК**.

Если требуется применить фильтр только по одному или нескольким значениям из длинного списка, сначала снимите флажок **Выделить все**, а затем выберите нужные значения.

Чтобы отфильтровать пустые значения (пустое значение означает отсутствие данных) в текстовых и числовых полях, а также в полях дат, снимите флажок **Выделить все**, а затем установите флажок значения **Пустые**.

Алгоритм применения фильтра по выделению:

1) откройте таблицу, запрос, форму или отчет в режиме таблицы, формы, отчета или макета;

2) убедитесь, что представление еще не отфильтровано. В области маркировки проверьте наличие значка **Без фильтра** или затененного значка **Нет фильтра**;

3) перейдите к записи, в которой содержится значение, используемое в качестве компонента фильтра, и щелкните внутри столбца (в режиме конструктора) или элемента управления (в режиме формы, отчета или макета).

Чтобы применить фильтр по частично выделенному значению, выделите нужные символы, на вкладке **Главная** в группе **Сортировка и фильтр** выберите команду Выделение и укажите фильтр, который требуется применить.

Алгоритм применения фильтра с помощью заполнения формы:

1) откройте таблицу или запрос в режиме таблицы или форму в представлении формы;

2) убедитесь, что представление еще не отфильтровано. В области маркировки проверьте наличие значка **Без фильтра** или затененного значка **Нет фильтра**;

3) на вкладке **Главная** в группе **Сортировка и фильтр** нажмите кнопку **Дополнительно** и выберите в контекстном меню команду **Фильтр по форме;**

4) выполните действия, соответствующие выбранному режиму. В **режиме Таблицы** щелкните первую строку в столбце, к которому нужно применить фильтр. Щелкните появившуюся стрелку и выберите значение. Чтобы добавить дополнительные значения, откройте вкладку **Или** в нижней части таблицы и выберите другое значение. В **режиме Формы** щелкните стрелку в элементе управления и выберите значение для фильтра. Чтобы добавить дополнительные значения, откройте вкладку **Или** в нижней части формы и выберите другое значение;

5) иногда может возникнуть необходимость в определении двух различных наборов условий (например составить список с именами контактов, проживающих в Казахстане, и контактов, имеющих дни рождения в апреле).

Чтобы получить все записи, соответствующие любому из нескольких наборов условий, введите первый набор условий. Затем откройте вкладку **Или** и введите следующий набор условий. Обратите внимание на то, что если значение поля нужно использовать в качестве фильтра независимо от других значений полей, необходимо ввести это значение на вкладке **Найти** и на каждой вкладке **Или**. Иначе говоря, вкладка **Найти** и каждая вкладка **Или** задают отдельный набор значений для фильтра.

Обратите внимание, что каждый раз, когда на вкладку **Или** добавляется условие, создается другая вкладка **Или**, что позволяет указать несколько альтернативных условий отбора.

Чтобы удалить фильтр и отобразить все записи, нажмите кнопку **Переключить фильтр** еще раз.

Чтобы изменить фильтр по форме, нажмите кнопку **Дополнительно** и еще раз выберите команду **Фильтр по форме**. Будет отображен текущий набор условий фильтра.

Алгоритм применения расширенного фильтра:

1) откройте таблицу, запрос, форму или отчет в режиме таблицы, формы, отчета или макета. Убедитесь, что представление еще не отфильтровано. В строке переходов по записям проверьте, что значок **Нет фильтра** отображается затененным (недоступен). Если строка переходов по записям не отображается, нажмите кнопку **Дополнительно** в группе **Сортировка и фильтр** на вкладке **Главная** и затем выберите команду **Очистить все фильтры** (если команда **Очистить все фильтры** недоступна, никакие фильтры не применены);

3) на вкладке **Главная** в группе **Сортировка и фильтр** нажмите кнопку **Дополнительно** и выберите в контекстном меню команду **Расширенный фильтр**.

4) добавьте в сетку поля, к которым требуется применить фильтр;

5) в строке **Условия отбора** укажите условие для каждого поля. **Условия** применяются в виде набора, и отображаются только записи, которые соответствуют всем условиям в строке **Условия отбора**. Чтобы указать альтернативные условия для отдельного поля, введите первое условие в строке Условия отбора, второе условие в строке **Или** и т. д. Любое условие, которое должно применяться в обоих наборах условий, необходимо ввести как в строку **Условия отбора**, так и в строку **Или.**

6) нажмите кнопку **Переключить фильтр**, чтобы увидеть отфильтрованные строки.

Чтобы научиться задавать условия, примените к представлению обычный фильтр или фильтр по выделенному, который позволяет получить желаемый результат. Затем откройте вкладку объекта **Фильтр.**

Удаление с экрана лишних данных

Для удаления с экрана (но не из таблицы) лишних данных (полей), а также выполнения операции копирования и вставки для столбцов, не являющихся соседними:

– одного столбца – установите в него курсор и вызовите правой клавишей контекстное меню. Выберите команду **Скрыть поля;**

– нескольких столбцов, а также восстановления их отображения – выберите команду **Отобразить поля.**

При этом, скрытие столбцов в режиме таблицы не делает скрытым поле в режиме **Конструктор.**

Значения, находящиеся в скрытых столбцах, могут быть использованы в условиях отбора фильтра. При этом столбец остается скрытым после применения фильтра.

Для сохранения изменений отображения столбцов следует выбрать команду **Сохранить** (вкладка **Файл**).

Фиксация столбцов

Для того чтобы зафиксировать столбцы, которые не будут уходить за край экрана при прокрутке необходимо:

1) выделите столбцы, которые необходимо зафиксировать;

2) правой клавишей мыши откройте контекстное меню и выберите команду З**акрепить поля.**

Новые зафиксированные столбцы добавляются справа к зафиксированным ранее.

Для отмены фиксации столбцов следует выбрать команду **Отменить закрепление всех полей**.

Создание простого отчета

Для автоматического создания простого отчета необходимо:

1) в **Области навигации** выберите нужную таблицу;

2) на вкладке **Создание** в группе **Отчеты** выберите кнопку **Отчет**. Будет сформирован отчет по выбранной таблице.

Задание для самостоятельного выполнения

1. Откройте БД Продажи, созданную в результате задания 1.

2. Введите данные в таблицы Склад, Фирмы, Товары. Данные к заполнению произвольны. Пример приведен в табл. 6-8.

Таблица 6 – Данные для заполнения таблицы «Склад»

№склада	Телефон	Адрес	Заведующий
10	8(495)953-0189	ул. Свободы,37	Иванов П.П.
20	8(499)712-2222	ул. Мелитопольская, 35	Гривко Л. К.
30	8(495)901-7044	ул. Кустарная, 28	Соломоник К.Ф.
40	8(495)359-0014	ул. Перерва, 2	Андреев П.Н.

Таблица 7 – Данные для заполнения таблицы «Фирмы»

Код фирмы	Название	Адрес	Телефон
100	Мир	Чонгарский б-р, 16	8(495) 152-4001
200	М. Видео	Маросейка, 6/8	8(495) 923-2906
300	ДиалЭлектроникс	Новослободская, 14/19	8(495) 978-1693

Таблица 8 – Данные для заполнения таблицы «Товары»

Код товара	Наименование	Марка	Цена	№ склада	Количество	Описание
1	Телевизор	GoldStar CM-2180K	$459			
2	Телевизор	Philips 25PT9001	$1499			
3	Телевизор	Panasonic 25V50R	$765			
4	Телевизор	GoldStar CF-14E20B	$230			
5	Видеомагнитофон	Panasonic HS-800EE	$1400			
6	Видеомагнитофон	Philips VR-756	$450			
7	Видеокамера	Samsung VP-J55	$530			
8	Видеокамера	Sharp E37	$845			
9	Музыкальный центр	Panasonic DH32	$320			
10	Музыкальный центр	Sony MJ-L1	$1289			

3. Для таблицы «Товары» в режиме конструктора выберите в столбце **Тип данных** поля «Наименование» **Мастер подстановок...** и введите в один столбец фиксированный набор используемых в этом поле значений. Для поля «Номер склада» также воспользуйтесь мастером подстановок и указав таблицу «Склады», выберите соответствующее для подстановки поле.

4. Введите данные в таблицу, используя для полей «Название» и «№ склада» значения из выпадающего списка. При заполнении поля «№ склада», учитывайте условие, что на одном складе хранится только один вид товара (телевизор, видеокамера и т.п.).

Данные о количестве и описании товара заполните самостоятельно произвольными значениями.

5. Для полей внешнего ключа таблиц «Контактные лица» и «Продажи» в режиме конструктора в столбце «Тип данных» выберите **Мастер подстановок** и укажите значения каких полей и из каких таблиц будут использованы в этом поле. В таблице «Продажи» для поля «Код фирмы» выберите в качестве столбца подстановки поля «Код фирмы» и «Название» таблицы «Фирмы». Для поля «Код товара» – из таблицы «Товары» поля «Код товара», «Наименование» и «Марка». На вкладке **Подстановка** свойств поля «Код товара» установите число строк, выводимых в поле со списком, значение **Да** на вывод заглавий столбцов, подставляемых значений и ненулевую ширину для всех трех столбцов.

6. Самостоятельно заполните данными таблицы «Контактные лица» и «Продажи». Количество проданных товаров в таблице «Продажи» не должно превышать количество товаров, имеющихся на складе.

7. Проверьте схему данных.

8. Откройте таблицу «Товары».

9. Добавьте запись: Видеокамера, Panasonic NV-DX1E, $2599.

10. Осуществите замену названий GoldStar на LG Electronics.

11. Отсортируйте данные по цене.

12. Используя фильтрацию, выберите данные: по названиям, например, о музыкальных центрах и по цене меньше определенного числа, например, меньше 800.

13. Отобразите на экране только данные полей «Наименование», «Марка» и «Цена», удалив с экрана лишние данные.

14. Добавьте поле «Изображение» (тип OLE).

15. Добавьте в первую запись объект – рисунок телевизора, созданный в графическом редакторе.

16. Пользуясь буфером обмена, скопируйте полученные данные в три последующие клетки.

17. На основе таблицы Товары, подготовьте простой отчет.

3 Создание запросов

Использование запросов позволяет осуществлять различные формы доступа к одной и той же информации.

Запрос – это объект БД, допускающий многократное использование.

Результат запроса – представленный в табличном виде набор данных. Запросы могут быть созданы как с помощью мастера запросов, так и самостоятельно, с помощью конструктора запросов.

Для создания нового запроса в режиме конструктора необходимо:

1) на вкладке **Создание** в группе **Запросы** выберите кнопку **Конструктор запросов**;

2) в диалоговом окне **Добавление таблицы** укажите имена таблиц, по полям которых будет производиться запрос, нажимая кнопку **Добавить** после каждого указанного имени таблицы;

3) нажмите кнопку **Закрыть.**

В специальном бланке запроса указываются условия отбора выводимых на экран полей и записей одной или нескольких таблиц и порядок их отображения. В бланке запроса содержится 6 строк.

MS Access позволяет выполнять *QBE-запросы* (запросы по образцу) и SQL-запрос.

QBE-запросы (QBE=Query By Example – запросы по образцу):

– запрос на выборку;

– перекрестный запрос;

– запрос на создание таблицы;

– запрос на обновление;

– запрос на добавление записей;

– запрос на удаление записей.

Каждый из этих типов указывается в дополнительной вкладке **Работа с запросами** группа **Тип запроса.**

Запросы SQL (Structured Query Language – Структурированный язык запросов). SQL – стандартизированная форма составления запросов для обработки реляционных баз данных. При выполнении QBE-запросов они транслируются в соответствующие SQL-запросы.

Запрос на выборку

Запрос на выборку является самым распространенным типом запроса. Данный запрос определяет, какие записи или поля из одной или нескольких таблиц будут отображены при его выполнении.

Для выбора записей, удовлетворяющих определенным критериям необходимо провести ряд действий:

1) в строке **Поле** щелкните в правой части клетки на стрелке, указывающей вниз и выберите имя поля, по которому будет осуществляться запрос. Если запрос осуществляется по полям из разных таблиц, то сначала щелкните в строке **Таблица** и укажите нужную таблицу, что позволит ограничить список полей в строке **Поле**. Если запрос будет осуществляться по нескольким полям, отобразите их имена в свободных клетках строки **Поле;**

2) проследите, чтобы в строке **Вывод на экран** флажок отображался бы галочкой;

3) в строке **Условие отбора** введите критерии выбора (для задания диапазона значений могут быть использованы операторы > (больше), >= (не менее), < (меньше), <= (не более) и Between (между) Выражение 1 and Выражение как с текстовыми и числовыми полями, так и с полями дат). Для ввода условия выборки можно использовать окно **Построитель выражений** (группа **Настройка запроса** – кнопка **Построить**);

4) если это нужно, сохраните запрос для последующего использования. Для выполнения запроса нажмите кнопку с восклицательным знаком **Выполнить** группа **Результаты**.

Сортировка блоков данных в запросе

Блоки данных в запросе могут быть рассортированы алфавитным или числовым способом в возрастающей (А-Я, 0-9) или убывающей (Я-А, 9-0) последовательности по содержимому отдельных полей. Можно одновременно производить сортировку по содержимому нескольких полей (до десяти):

1) щелкните мышью в строке **Сортировка** того столбца (поля), по которому необходимо произвести сортировку;

2) укажите способ сортировки.

Запрос с параметром (параметрический запрос)

Как правило, запросы с параметром создаются в тех случаях, когда предполагается выполнять этот запрос многократно, изменяя лишь условия отбора. В отличие от запроса на выборку, где для каждого условия отбора создается свой запрос и все эти запросы хранятся в БД, параметрический запрос позволяет создать и хранить один единственный запрос и вводить условие отбора (значение параметра) при запуске этого запроса, каждый раз получая новый результат. Значение параметра задается в специальном диалоговом окне. В случае, когда значение выводимых данных должно быть больше или меньше указываемого значения параметра, в поле **Условие отбора** бланка запроса перед параметром, заключенным в квадратные скобки ставится соответствующий знак.

Можно также создавать запрос с несколькими параметрами, которые связываются друг с другом логическими операциями «И» и «ИЛИ». В момент запуска запроса на выполнение MS Access отобразит на экране диалоговое окно для каждого из параметров. Помимо определения параметра в бланке запроса, необходимо указать с помощью кнопки **Параметры** (группа **Показать или скрыть**) соответствующий ему тип данных.

Последовательность действий при создании параметрического запроса:

1) откройте в режиме конструктора окно запроса и добавьте в него таблицу. Создайте запрос, «перетащив» необходимые поля в бланк запроса и задав условие выбора;

2) в качестве условия введите параметр, заключенный в квадратные скобки (например, «[Введите название]» или «>[Выше какого роста?]»).

3) выберите команду **Параметры** (меню **Запрос**);

4) в окне **Параметры запроса** введите без квадратных скобок параметр и укажите соответствующий ему тип данных. Нажмите **ОК**.

5) нажмите кнопку **Выполнить** (группа **Результаты**);

6) в появившемся окне укажите значение параметра;

7) результат запроса будет содержать только те записи, которые удовлетворяют заданному значению параметра.

Вычисления в запросах

Запрос можно использовать для выполнения расчетов и подведения итогов из исходных таблиц.

Для создания вычисляемых полей используются математические и строковые операторы. При этом Access проверяет синтаксис выражения и автоматически вставляет следующие символы:

– квадратные скобки [], в них заключаются имена элементов управления;

– знаки номеров (#), в них заключаются распознанные даты;

– кавычки (""), в них заключается текст, не содержащий пробелов или знаков пунктуации.

Выражения, определяемые пользователем, дают возможность выполнять действия с числами, датами и текстовыми значениями в каждой записи с использованием данных из одного или нескольких полей. Например, обычное выражение позволяет найти разность значений двух полей типа даты, соединять несколько строковых значений в текстовом поле или умножить значения одного поля на итоговое значение.

Поле, содержимое которого является результатом расчета по содержимому других полей, называется *вычисляемым полем*. Вычисляемое поле существует только в результирующей таблице.

Общий формат вычисляемого поля выглядит так:

Имя вычисляемого поля: Выражение для создания вычисляемого поля

Примеры:

Прибыль:[Доход]-[Расход];

Цена со скидкой:[Цена]-[Цена]*0,1.

Для расчетов с использованием формул, определяемых пользователем, требуется создать новое вычисляемое поле прямо в бланке запроса путем простого ввода выражения для вычисления в ячейку **Поле** пустого столбца бланка запроса.

После выполнения запроса вычисляемое поле, основанное на этом выражении, выводит на экран результат вычислений, а не само выражение.

Последовательность действий построения выражения в запросе:

1) в строку **Поле** пустого столбца бланка запроса введите выражение, начинающееся со знака «=» и состоящее из имен полей, записанных в квадратные скобки и какой-либо арифметической или другой операции;

2) после выполнения запроса в результирующей таблице появится новое поле с названием «Выражение 1», используемым в качестве имени вычисления выражения;

3) в режиме конструктора запроса измените имя «Выражение 1» на более значимое.

Для того чтобы ввести сложные вычисления используйте окно **Построитель выражения**, которое вызывается нажатием кнопки **Построить** (группа **Настройка запроса**). Построитель выражений облегчает создание выражений, позволяя выбирать его составляющие элементы (арифметические операции, встроенные функции, названия полей имеющихся в БД таблиц и запросов и т.п.) при помощи кнопок и списков.

Результаты вычислений также могут быть использованы в условиях отбора для определения записей, которые выбираются в запросе, или для определения записей, над которыми производятся какие-либо действия.

Например, следующее выражение в ячейке строки **Условие отбора** позволяет отбирать в запросе только те записи, которые в поле Дата продажи имеют значение, попадающее в интервал между текущей датой и датой, отстоящей от нее на один месяц, т.е. данные за последний месяц:

Between Date() And DateAdd("m",1,Date())

Запросы позволяют производить итоговые вычисления. Для этих целей в Access предусмотрены статистические функции SQL. Статистическую функцию задают в строке **Групповая операция** бланка запросов, которая появляется при выполнении команды **Итоги** (группа **Показать и скрыть**). Заполняя ячейки в строке **Групповая операция**, можно выполнить расчеты для групп записей и вычислить сумму, среднее, количество или другой тип итогового значения для вычисляемого поле (табл. 9).

Таблица 9 – Групповые операции

Функция SQL	Действие
1	2
Sum	Суммирование значений определенного поля
Avg	Вычисление среднего значения данных определенного поля
Min	Вычисление минимального значения поля
Max	Вычисление максимального значения поля
Count	Вычисление количества записей, отобранных запросом по условию
First	Определяется первое значение в указанном поле записей, отобранных запросом

Продолжение табл. 9

1	2
Last	Определяется последнее значение в указанном поле записей, отобранных запросом
StDev	Вычисляется стандартное отклонение значений данного поля, для всех записей, отобранных запросом
Var	Вычисляется вариация значений данного поля для всех записей, отобранных запросом

Для выполнения запроса на итоговое вычисление необходимо:

1) находясь в режиме конструктора запроса, выберите команду **Итоги** (группа **Показать или скрыть**). В результате чего в бланке запроса появится строка **Групповая операция**;

2) для соответствующего поля выберите нужную функцию из списка.

Перекрестный запрос

Перекрестный запрос применяется в том случае, если необходимо объединить данные в формате строк-столбцов. В качестве заголовков для столбцов при проектировании таких запросов можно указать значения некоторых полей или выражений. Для его построения необходимо:

1) в режиме конструктора сформируйте запрос, добавив таблицу, которая должна лежать в его основе;

2) выберите команду **Перекрестный** (группа **Тип запроса**). Строка запроса **Вывод на экран** в бланке запроса изменится на новую строку **Перекрестная таблица** и перед ней появится строка **Групповая операция**;

3) в строке **Поле** укажите поле, значения которого в новой таблице должны появиться в виде строк; поле, значения которого в новой таблице должны появиться в виде столбцов и поле, содержимое которого в перекрестной таблице необходимо индицировать в качестве значения. Полей, которые будут использованы в качестве заголовков, может быть несколько;

4) щелкните мышью в строке **Перекрестная таблица** и выберите соответствующие значениям данных полей опции из разворачивающегося списка;

5) для поля, содержимое которого индицируется в качестве значений, в строке **Групповая операция** введите необходимую функцию, например, автосуммирования (Sum), определения среднего значения (Avg) или количества (Count).

На основе данных перекрестного запроса можно строить диаграммы, представленные в виде формы.

Задание для самостоятельного выполнения

1. Откройте БД «Продажи».

2. Создайте запрос для отображения названий товаров, их цен и телефонов складов, на которых они хранятся. Сохраните запрос, присвоив ему имя, отражающее смысл выполняемого запроса.

3. Создайте и сохраните запрос для отображения в алфавитном порядке дат продаж телевизоров с указанием их марок и проданного количества, а также названий, адресов и телефонов фирм, их закупивших.

4. Создайте и сохраните запрос для отображения в алфавитном порядке тех названий товаров, которые были проданы со скидкой, с указанием названий фирм-покупателей и закупленного количества.

5. Создайте и сохраните запрос для отображения наименований и марок товаров, проданных со скидкой с указанием цен со скидкой. В том случае, если скидка не предусмотрена – указать цену без изменения.

6. Создайте и сохраните запрос для отображения в алфавитном порядке фамилий, домашних и рабочих телефонов и адресов директоров фирм-покупателей.

7. Создайте и сохраните параметрический запрос для отображения товаров стоимостью до определенной суммы, названия и марки этого товара, а также его цены. Выполните его для нескольких значений параметра.

8. Создайте и сохраните параметрический запрос для отображения всех сведений о контактных лицах фирмы, определяемой значением параметра.

9. Создайте и сохраните запрос для отображения количества товаров, оставшихся на каждом складе, с указанием номера, адреса и телефона склада, наименования и марки товара.

10. Создайте и сохраните запрос для отображения средних цен на все товары.

11. Создайте и сохраните перекрестный запрос, отображающий количество всех товаров, проданных разным фирмам, с указанием наименований товаров в заголовках строк и указанием названий фирм в заголовках столбцов.

4 Запросы на изменение структуры данных базы

Запрос на создание таблицы

БД на физическом уровне хранит только таблицы. Набор записей запросов физически не существует в БД. Access создает его из данных таблиц только во время выполнения запроса. Иногда возникает необходимость сохранить извлекаемые с помощью запроса на выборку данные в новой таблице. Тогда алгоритм действий следующий:

1) создайте новый запрос на выборку и проверьте его корректность, перейдя в режим **Таблица**. Для создания резервной копии таблицы (таблицы, содержащей те же поля и в том же количестве, что и в оригинале), чтобы не перетаскивать все поля таблицы в строку **Поле**, достаточно поместить туда из начала списка полей таблицы символ *, заменяющий все поля таблицы;

2) преобразуйте запрос на выборку в запрос на создание новой таблицы. Для этого, в группе **Тип запроса**, выберите команду **Создание таблицы**;

3) в появившемся окне введите имя новой таблицы и нажмите **ОК;**

4) выполните запрос.

Запрос на обновление

Используя этот тип запроса, можно изменить в базовой таблице группу блоков данных, отобранную на основе определенных критериев:

1) создайте новый запрос на выборку и проверьте его корректность, перейдя в режим **Таблица**;

2) преобразуйте запрос на выборку в запрос на обновление. Для этого, вернувшись в режим конструктора, выберите команду **Обновление** (группа **Тип запроса;**

3) в появившейся в бланке запроса строке **Обновление** в соответствующих столбцах задайте новые значения полей таблицы. В качестве таковых могут выступать и вычисляемые значения. В случае необходимости воспользуйтесь построителем выражений;

4) выполните запрос.

Запрос на добавление записей

С помощью этого типа запроса блоки данных одной таблицы (все или отобранные запросом) можно присоединить в конец другой таблицы, что реализуется следующим образом:

1) создайте новый запрос на выборку тех блоков данных, которые будут добавлены в некоторую таблицу и проверьте его корректность, перейдя в режим таблицы;

2) преобразуйте запрос на выборку в запрос на добавление. Для этого, вернувшись в режим конструктора, выберите команду **Добавление** (группа **Тип запроса**);

3) в появившемся окне введите имя таблицы, к которой нужно присоединить данные и нажмите **ОК;**

4) выполните запрос.

Запрос на удаление записей

С помощью данного типа запроса можно удалить из базовой таблицы группу блоков данных, отобранных по определенным критериям. При этом следует тщательно проанализировать критерии отбора, поскольку эту операцию нельзя отменить. Алгоритм следующий:

1) создайте новый запрос на выборку удаляемых блоков данных. Отбор блоков данных выполняется в соответствии с заданными в строке **Условие** критериями;

2) проверьте корректность сформулированных условий, перейдя в режим таблицы;

3) преобразуйте запрос на выборку в запрос на удаление записей. Для этого, вернувшись в режим конструктора, выберите команду **Удаление** (группа **Тип запроса**).

4) в появившейся строке **Удалить** установите критерии отбора;

5) выполните запрос.

Задание для самостоятельного выполнения

1. Откройте базу данных, созданную в ходе выполнения предыдущих упражнений.

2. Создайте и сохраните запрос на создание резервной копии таблицы «Товары». Присвойте ей имя «Товары 1».

3. Создайте и сохраните запрос на обновление в таблице «Товары 1» цен с учетом сезонных скидок в 10%.

4. Создайте и сохраните запрос на обновление в таблице «Товары 1» количества товара, оставшегося на складе после продаж.

5. Создайте и сохраните запрос на создание таблицы «Видеокамеры», отображающей данные о ценах на видеокамеры, марке товара, а также о названиях и телефонах фирм, их реализующих.

6. Создайте и сохраните запрос на добавление в таблицу «Видеокамеры» данных о видеомагнитофонах.

7. Используя команду **Заменить** (вкладка **Главная** группа **Найти**), измените имя таблицы «Видеокамеры» на «Видеотовары».

8. Создайте и сохраните запрос на удаление данных о видеомагнитофонах Sony E150EE из таблицы «Видеотовары».

5 Создание форм и отчетов

Создание формы

Формы Access позволяют создавать пользовательский интерфейс для таблиц базы данных. Хотя для выполнения тех же самых функций можно использовать режим таблицы, формы предоставляют преимущества для представления данных в упорядоченном и удобном для пользователя виде:

– форма представляет собой некий электронный бланк, в котором имеются поля для ввода данных;

– в форме каждое поле можно разместить в точно заданном месте, выбрать для него цвет и заливку;

– форму можно помещать вычисляемые поля;

– OLE-обьекты можно увидеть только в форме или отчете;

– в форме намного проще работать с большими текстами поля типа MEMO в текстовом окне с полосами прокрутки;

– форма строится на основе таблицы или запроса. При каждом открытии сохраненной формы обновляются данные запроса, на основе которого создается форма. Благодаря этому содержимое формы всегда соответствует информации в таблицах и запросах;

– формы могут быть выведены на экран в трех видах: режим формы, режим макета и режим конструктора. Для перехода из одного режима в другой используются команды группы **Режимы**.

MS Access предоставляет быстрый способ создания формы на основе таблицы с использованием **Мастера Форм**. Он задает пользователю вопросы о

структуре и оформлении формы. Результатом диалога пользователя и мастера форм является готовая к использованию форма.

Для создания формы в режиме конструктора необходимо:

1) в области навигации выберите таблицу, по которой будет создаваться форма;

2) на вкладке **Создание** в группе **Формы** выберите команду **Форма;**

3) выберите режим конструктора. При открытии окно конструктора содержит три области: заголовок формы, область данных, примечание формы.

Поля, размещенные в области данных, состоят из надписи поля и поля для ввода данных. Если выделить надпись или само поле, то ко второму элементу автоматически добавляется манипулятор перемещения и можно перемещать их в паре или по отдельности. В случае, когда нет необходимости в выводе надписи поля рядом с самим полем, удалить ее можно следующим образом: выделить объект **Надпись** и нажать клавишу **Delete.**

Формы для связанных таблиц

В таких формах можно одновременно отобразить информацию из двух (или более) связанных таблиц. Кроме того, такая форма позволяет выполнить редактирование данных, содержащихся в обеих таблицах.

В результате создания этой формы на экране выводятся только те записи подчиненной таблицы, которые связаны с текущей записью исходной (главной) таблицы:

1) выберите команду **Мастер форм** (вкладка **Создание** группа **Формы**);

2) в появившемся диалоговом окне укажите имена полей для главной и подчиненной форм и порядок их размещения в новой форме, выбрав имя таблицы из раскрывающегося списка **Таблицы/Запросы**. Нажмите **Далее;**

3) в следующем окне выберите переключатель **Подчиненные формы**

4) выберите вид подчиненной формы;

5) озаглавьте главную и подчиненную формы и нажмите кнопку **Готово.**

Для просмотра записей главной формы используются кнопки просмотра в нижней части окна. Выше нее выводится строка для просмотра записей подчиненной формы, которые представлены в виде таблицы.

Создание отчета

Располагая базой данных можно распечатать любую таблицу, запрос или форму. Однако результаты печати не будут презентабельно, так как эти инструменты не предназначены для печати. В Access 2010 отчет представляет собой форму специального типа, предназначенную для вывода на печать. Но в отличие от форм отчеты не предназначены для вывода в окне и предназначены только для печати, т.е. создают не экранные, а печатные документы.

При создании отчета Access всегда оперирует данными только одной таблицы или запроса. Если необходимо объединить информацию из нескольких таблиц и (или) запросов в одном отчете, то прежде следует собрать желаемые данные в новом запросе.

Для создания отчета с помощью **Мастера Отчетов** необходимо:

1) на вкладке **Создание** группа **Отчеты** выберите команду **Мастер отчетов;**

2) укажите имя таблицы или запроса, на основе которых создаете отчет;

3) выберите поля, данные которых будут помещены в отчет;

4) задайте требуемый порядок сортировки полей;

5) выберите вид макета отчета;

6) Задайте имя отчета;

7) нажмите кнопку **Готово**.

Для создания отчета в режиме **Конструктора** необходимо:

1) на вкладке **Создание** группа **Отчеты** выберите команду **Конструктор отчетов**;

2) на вкладке **Конструктор** группа **Сервис** выберите команду **Добавить поля**;

3) укажите имя таблицы, на которой должен базироваться отчет и выберите поля, данные которых будут отображаться в отчете.

Структурные элементы отчета:

– заголовок отчета – печатается только в начале отчета, используется на титульной странице;

– верхний колонтитул – печатается вверху каждой страницы;

– заголовок группы – печатается перед обработкой первой записи группы;

– область данных – печатается каждая запись таблицы или динамического набора данных запроса;

– примечание группы – печатается после обработки последней записи группы;

– нижний колонтитул – печатается внизу каждой страницы;

– примечание отчета – печатается в конце отчета после обработки всех записей.

Проектирование отчета состоит в создании структуры его разделов и в размещении элементов управления внутри этих разделов, а также в задании связей между этими элементами и полями таблиц или запросов базы данных.

Отчеты предназначены для вывода информации на принтер, поэтому для расчета расположения данных на печатной странице программа Access 2010 должна «знать» все необходимое об особенностях принтера. Эти данные Access получает от операционной системы. Соответственно, принтер в системе должен быть установлен.

Создание почтовых наклеек

Для переписки, как правило, используются почтовые наклейки и стандартные письма, обычно называемые составными документами рассылки. В Access почтовые наклейки создаются с помощью отчетов. Подобно любому другому отчету, отчет для создания наклеек состоит из элементов управления. Для создания наклеек лучше всего использовать команду **Наклейки** (вкладка **Создание** группа **Отчеты**). Изменение макета наклейки следует производить в

режиме конструктора. Печатать почтовые наклейки можно непосредственно из окна предварительного просмотра.

Создание элементов формы или отчета

Как в формах, так и в отчетах помимо информации из БД можно отображать и дополнительную информацию. Окно формы может содержать следующие элементы: подписи, поля, поля со списком, списки, выключатели, переключатели, флажки и кнопки. Кроме того, форму (отчет) можно дополнить иллюстрацией (рисунком или диаграммой), текстом и линиями различного типа. Для оформления форм (отчетов) также может быть использована возможность изменения начертания, стиля и выравнивания данных, которые отображаются в полях, а также цвета символов, фона и границы.

Создание элементов окна осуществляется в режиме **Конструктора.**

Существует три основных типа элементов управления: присоединенные, свободные, вычисляемые.

Присоединенные элементы управления – элементы, связанные с полем таблицы. При вводе значения в присоединенный элемент управления поле таблицы в текущей записи автоматически обновляется. Большинство элементов управления, в том числе объекты OLE, можно присоединить к полю.

Чаще всего присоединенные элементы управления содержат данные текстового типа, а также даты, числа, логические данные (Да/Нет), рисунки и поля MEMO.

Свободные элементы управления сохраняют введенную величину, не обновляя при этом поля таблицы. Их можно использовать для отображения: текста; значений, которые должны быть переданы макросам; линий и прямоугольников. Кроме того, их можно использовать для хранения объектов OLE (например, рисунков), которые расположены не в таблице, а в самой форме.

Свободные элементы управления называют также переменными или переменными памяти.

Вычисляемые элементы управления создают на основе выражений, например, функций или формул. Поскольку они не присоединены к полям таблицы, они не обновляют содержание полей таблицы. Этот элемент управления позволяет производить необходимые вычисления, используя данные полей таблицы, с последующим отображением в форме.

Рассмотрим назначение основных элементов панели инструментов:

– *выбор объектов* – позволяет изменить указатель курсора на инструмент выбора объекта;

– *мастера элементов* – позволяет включать и отключать мастера по созданию элементов управления;

– *надпись* – предназначена для вывода на экран не изменяющегося текста, например, заголовков, подписей или пояснений. Надпись относится к свободным элементам управления, в которые нельзя вводить данные;

– *поле* – позволяет создать область для отображения, ввода или изменения данных. В поле можно использовать данные любого типа: текст, числа, дата/время, логические величины и МЕМО. Поля могут быть как присоединенными, так и свободными. В них можно использовать поля из таблиц или запросов, а также вычисляемые выражения, поэтому такие элементы управления называют связанными полями. При создании связанного поля вместе с ним одновременно образуется еще один элемент управления - присоединенная надпись;

– *группа параметров* – позволяет создать область настраиваемого размера для размещения набора флажков, переключателей или выключателей, представляющих набор альтернативных значений;

– *выключатель* – позволяет создать кнопку, связанную с логическим полем. Элемент может находиться в двух состояниях: «Истина» – кнопка нажата, «Ложь» – кнопка отжата;

– переключатель — предназначен для создания кнопки (называемой радиокнопкой). Ее функции аналогичны функциям выключателя. Элемент находится в двух состояниях: «Истина» – кружок с точкой, «Ложь» – пустой

кружок. С кнопкой можно связать команды, например, выполняющие фильтрацию;

– *флажок* – предназначен для создания флажка связанного с логическим полем. Действуют аналогично переключателям, но в отличие от них, допускают множественный выбор. Элемент может находится в двух состояниях: «Истина» – квадрат с галочкой, «Ложь» – пустой квадрат;

– *поле со списком* – позволяет создать составной элемент управления, объединяющий поле и раскрывающийся список значений. Для ввода значения, можно ввести значение в поле или выбрать значение в списке;

– *список* – позволяет создать список, допускающий прокрутку, и предназначенный для выбора значения. Позволяет отображать список значений в форме или отчете. В списках можно также отображать заголовки столбцов;

– *кнопка* – позволяет создать кнопку, используемую для выполнения набора макрокоманд Access или процедур VBA;

– *рисунок* – позволяет создать рамку, в которой в форме или отчете выводится неизменяемый рисунок. Поскольку рисунок не является объектом OLE, то после помещения рисунка в форму или отчет не допускается его изменение из Microsoft Access;

– *свободная рамка объекта* – позволяет создать рамку для отображения в форме или отчете объектов OLE, как правило, набор иллюстраций или диаграмму. Рамка не связана ни с каким полем таблиц базы данных;

– *присоединенная рамка объекта* –для отображения в форме или отчете объектов OLE, таких как набор иллюстраций или диаграммы. С присоединенной рамкой связано одно из полей таблиц. При переходе от записи к записи в форме или отчете выводятся разные объекты;

– *конец страницы* – позволяет создать элемент управления, указывающий принтеру начало новой страницы в печатной форме или новой страницы в отчете. Этот элемент управления не появляется в форме или запросе в режиме формы;

– *вкладка* – позволяет вставить элемент управления «Вкладка» для создания вложенных форм. Страницы элемента управления «Вкладка» могут содержать другие элементы управления;

– *подчиненная форма/отчет* – предназначена для добавления в основную форму или основной отчет подчиненной формы или подчиненного отчета соответственно. Добавляемые подчиненная форма или подчиненный отчет должны существовать;

– *линия* – позволяет создать прямую линию, которую можно перемещать и размеры которой можно изменять. Цвет и толщину линии можно изменить с помощью кнопок панели инструментов «Панель форматирования» или окна свойств. Используется для разделения элементов формы или отчета;

– *прямоугольник* – позволяет создать прямоугольник, который можно перемещать и размеры которого можно изменять. Используется для выделения элементов формы;

– *дополнительные элементы* – выбор этой кнопки открывает список дополнительных элементов управления ActiveX, которые можно использовать в формах и отчетах.

Для создания элемента управления: текста, поля, линии, прямоугольника (рамки), кнопки и др. необходимо:

1) щелкните на соответствующей пиктограмме;

2) укажите курсором мыши (крест с уменьшенным изображением создаваемого элемента) место для создаваемого элемента.

После того, как будет отпущена кнопка мыши для создания некоторых элементов (таких как, например, поле со списком или кнопка) Access выводит на экран **Мастер**. Так, после создания кнопки появляется **Мастер**, предлагающий выбрать тип действия, которое будет привязано к этой кнопке (переходы между записями, работа с формой или другие типы, например, работа с запросами в пункте «Разное»).

Внешний вид, структура и режимы работы отдельных управляющих элементов определяются значениями характеристик этих объектов (кнопка **Страница свойств** группа **Сервис**).

Добавление вычисляемых выражений в формы и отчеты

Для выполнения добавления вычисляемых выражений в формы и отчеты необходимо выполнить следующие действия:

1) откройте форму (отчет) в режиме конструктора;

2) выберите кнопку **Поле** (группа **Элементы управления**);

3) выберите мышью пустое место в любой области формы или отчета (например, область примечаний);

4) для появившегося нового поля укажите необходимые свойства: откройте окно свойств поля (группа **Сервис** кнопка **Страница свойств**);

5) для свойства «Данные» введите начиная со знака «=» нужное выражение, заключив имена полей БД в квадратные скобки. В качестве выражения может быть использована как встроенная функция (например, «=DATE()» – системная дата), так и любое действие над значениями полей с использованием арифметических или других операций;

6) для свойства «Формат поля» выберите из списка тип вычисляемых данных.

7) в случае необходимости вставьте рядом с полем элемент **Надпись** и заполните его нужным текстом.

8) Перейдите в режим **Формы (Отчета)**.

Задание для самостоятельного выполнения

1. Откройте базу данных «Продажи», созданную в результате выполнения предыдущих упражнений.

2. С помощью мастера форм создайте и сохраните форму на основе таблицы «Товары», выводящую в один столбец значения полей «Наименование», «Марка», «Цена», «Описание».

3. В созданную форму добавьте кнопки, позволяющие осуществлять переход между записями.

4. Создайте форму для таблиц «Фирмы» и «Контактные лица», отображающую данные о сотрудниках каждой фирмы, через которых осуществляется реализация товаров, используя таблицу «Фирмы» в качестве главной, а таблицу «Контактные лица» в качестве подчиненной.

5. Создайте форму для таблиц «Склады» и «Товары», отображающую данные о товарах, хранящихся на каждом складе, используя таблицу «Склады» в качестве главной, а таблицу Товары в качестве подчиненной.

6. Создайте запрос и на его основе форму для таблицы «Товары 1», отображающую данные о ценах на телевизоры, их марку, изображение и описание, включив в область примечаний минимальную цену.

7. Добавьте в созданную форму кнопку, при нажатии на которую будет выполняться запрос на обновление данных, созданный в третьем пункте практического задания № 4.

8. Создайте отчет по запросу для таблиц «Товары» и «Поставщики», отображающий данные о музыкальных центрах: их марку и цену, а также название фирмы-поставщика, включив в область заголовка соответствующее название отчета, в область верхнего колонтитула системную дату, а в область примечаний – среднюю цену (предварительно создав запрос по интересующим критериям). Оформите отчет с помощью элементов рисования панели элементов. В случае необходимости воспользуйтесь информацией **Помощника по разработке отчета.**

6 Макросы и макрокоманды

Создание макроса пользовательского интерфейса

В Microsoft Access 2010 макросы, связанные с объектами пользовательского интерфейса (такими как кнопки, текстовые поля, формы и

отчеты), называются *макросами пользовательского интерфейса.* Этим они отличаются от макросов данных, которые связываются с таблицами.

С помощью макросов пользовательского интерфейса можно автоматизировать последовательности действий, таких как открытие другого объекта, применение фильтра, запуск операции экспорта и многие другие задачи.

Макросы могут содержаться в объектах макроса (иногда их называют изолированными макросами) либо могут быть внедрены в свойства событий форм, отчетов или элементов управления. Внедренные макросы становятся частью объекта или элемента управления.

Объекты макроса отображаются в области навигации в группе **Макросы**; внедренные макросы не отображаются.

Каждый макрос состоит из одной или нескольких макрокоманд. В зависимости от текущего контекста некоторые макрокоманды могут быть недоступны. В частности, при работе с веб-базой данных недоступны макрокоманды, несовместимые с функцией публикации в службах Access.

Создание изолированного макроса

Рассмотрим процедуру создания объекта изолированного макроса, который будет отображаться в разделе **Макросы** в области навигации. Изолированные макросы удобно использовать повторно в разных частях приложения. Вызывая тот или иной макрос из других макросов, можно избежать дублирования программного кода в нескольких местах. Процедура создания макроса:

1) на вкладке **Создание** в группе **Макросы и код** нажмите кнопку **Макрос**;

2) в приложении Access откроется конструктор макросов;

3) на панели быстрого запуска нажмите кнопку **Сохранить**;

4) в диалоговом окне **Сохранить как** введите имя макроса, а затем нажмите кнопку **ОК**;

5) перейдите к разделу **Добавление команд в макрос**.

Создание внедренного макроса

В описанной ниже процедуре создается макрос, который внедряется в свойство события объекта. Такой макрос не отображается в области навигации, однако его можно вызывать из событий, таких как **Загрузка** или **Нажатие кнопки**. Поскольку макрос становится частью объекта формы или отчета, внедренные макросы рекомендуется создавать для автоматизации задач, которые специфичны для определенной формы или отчета. Алгоритм реализации следующий:

1) в области навигации щелкните правой кнопкой мыши форму или отчет, которые будут содержать макрос, и выберите пункт **Режим макет;**

2) если окно свойств не открыто, нажмите клавишу F4;

3) выберите элемент управления или раздел, содержащий свойства события, в который нужно встроить макрос. В верхней части окна свойств из выпадающего списка в разделе **Тип выбора** можно выбрать элемент управления или **раздел**, а также **форму** или **отчет** целиком;

4) в области задач «Страница свойств» откройте вкладку **Событие;**

5) щелкните поле **Свойства** для события, которое должно запускать макрос. Например, если требуется запускать макрос при нажатии кнопки, щелкните поле **Нажатие кнопки.**

Примечания:

– если поле **Свойства** содержит слова «[Внедренный макрос]», это означает, что для данного события уже создан макрос. Если нужно отредактировать его, выполните остальные действия процедуры;

– если поле **Свойства** содержит слова «[Процедура обработки событий]», это означает, что для данного события уже создана процедура Visual Basic для приложений. Прежде чем встраивать в это событие макрос, необходимо удалить процедуру. Чтобы сделать это, удалите слова «[Процедура обработки событий]», однако сначала следует просмотреть соответствующую процедуру и убедиться, что ее удаление не нарушит функциональность базы данных. Иногда

функциональность процедуры VBA можно воссоздать с помощью внедренного макроса:

1) нажмите кнопку **Построение …;**

2) в появившемся диалоговом окне **Построитель** установите флажок **Конструктор макросов** и нажмите кнопку **ОК**.

В Access запустится конструктор макросов. Перейдите к процедуре добавления команд в макрос.

Добавление команд в макрос

Команды представляют собой отдельные действия, составляющие макрос, и имя каждой из них соответствует выполняемому ей действию (например, **НайтиЗапись** или **ЗакрытьБазуДанных**).

Действие 1. Выбор или поиск макрокоманды.

Первым действием при добавлении команды является ее поиск в раскрывающемся списке **Добавить новую макрокоманду** или в каталоге макрокоманд.

Примечания:

– по умолчанию в раскрывающемся списке **Добавить новую макрокоманду** и каталоге макрокоманд выводятся только те команды, которые можно выполнить в недоверенных базах данных. Чтобы увидеть полный список команд на вкладке **Конструктор** в группе **Показать или скрыть** выберите пункт **Показать** все действия;

– если каталог макрокоманд не отображается, на вкладке **Макет** в группе **Показать/скрыть** нажмите кнопку **Каталог макрокоманд.**

Чтобы найти макрокоманду, воспользуйтесь одним из способов:

– щелкните стрелку в раскрывающемся списке **Добавить новую макрокоманду** и прокрутите список до нужной команды. В верхней части списка выводятся элементы программного потока, а затем в алфавитном порядке – макрокоманды;

– перейдите к нужной команде в области каталога макрокоманд. Команды группируются по категориям. Чтобы просмотреть команды в той или иной

категории, разверните ее. При выборе команды в нижней части каталога макрокоманд появляется ее краткое описание;

– найдите нужную команду в каталоге макрокоманд, введя запрос в поле поиска в верхней части соответствующей области. По мере ввода список действий фильтруется, при этом отображаются все макросы, содержащие введенный текст. В Access поиск текста выполняется как по именам макросов, так и по их описаниям.

Действие 2. Добавление макрокоманды в макрос.

Найдя нужную макрокоманду, добавьте ее в макрос одним из указанных ниже способов:

– выберите команду в списке **Добавить новую макрокоманду** или просто начните вводить ее имя в поле. Access добавит команду в то место, где отображается список **Добавить новую макрокоманду;**

– перетащите команду из каталога макрокоманд на область макроса. При этом появится полоса вставки, указывающая, в какое именно место будет вставлена команда, как только будет отпущена кнопка мыши;

– дважды щелкните команду в каталоге макрокоманд.

Если в области макроса выделена команда, Access добавит новую макрокоманду под выделенной.

Если в области макроса выделен блок **Группа**, **Если**, **Иначе если**, **Иначе** или **Вложенный макрос**, Access добавит новую макрокоманду в соответствующий блок.

Если в области макроса не выделена ни одна команда или блок, Access добавит новую макрокоманду в конец макроса.

Примечания:

– все ранее созданные макросы выводятся в узле **В этой базе данных** в каталоге макрокоманд.

– при перетаскивании изолированного макроса (указанного в разделе **Макросы**) в область макроса создается команда **ЗапускМакроса**, которая

запускает макрос, в который была перетащена команда. После этого из раскрывающегося списка можно запускать вложенные макросы (если они есть);

– чтобы просто скопировать команды из изолированного макроса в текущий (вместо создания команды **ЗапускМакроса**), щелкните его в каталоге макрокоманд и выберите команду **Добавить копию макроса**;

– при перетаскивании внедренного макроса (указанного в списке объекта формы или отчета) в область макроса действия из него копируются в текущий макрос;

– можно создать макрокоманду, перетащив объект базы данных из области навигации в область макроса. При перетаскивании таблицы, запроса, формы, отчета или модуля в область макроса приложение Access добавляет макрокоманду, открывающую таблицу, запрос, форму или отчет. При перетаскивании макроса добавляется макрокоманда, запускающая макрос.

Действие 3. Заполнение аргументов.

Большинству макросов необходим как минимум один аргумент. Чтобы просмотреть описания аргументов, выберите макрокоманду и наведите указатель на нужный аргумент. Для многих аргументов значение можно выбрать в раскрывающемся списке. Если аргумент требует ввода выражения, функция **IntelliSense** поможет ввести его, предлагая по мере ввода допустимые значения (рис. 53).

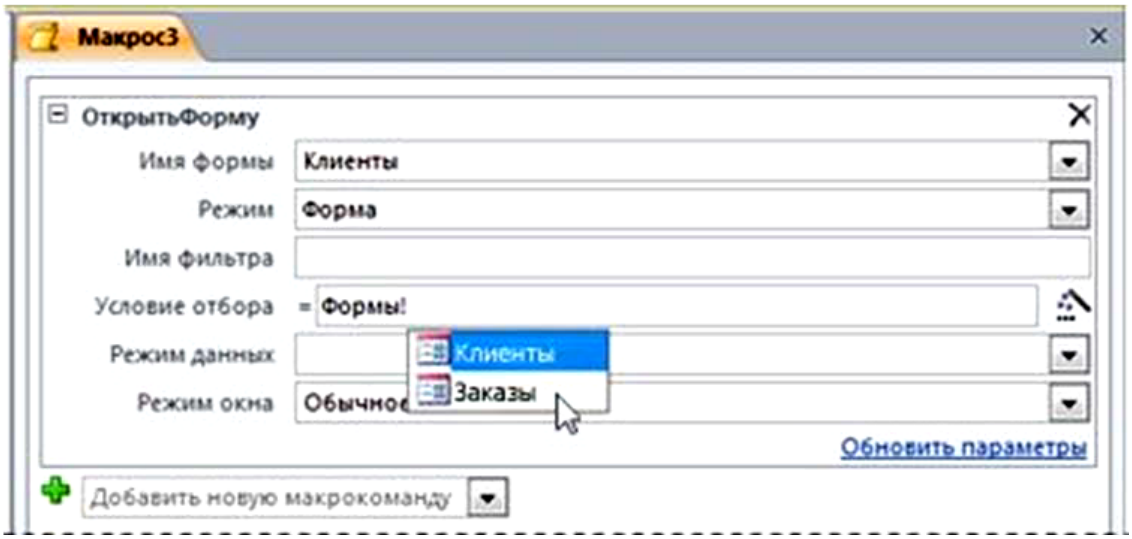


Рис. 53 – Заполнение аргументов макроса

Увидев нужное значение, добавьте его в выражение. Для этого дважды щелкните значение или выделите его с помощью клавиш со стрелками, а затем нажмите клавишу **Tab** или **Enter**.

Перемещение макрокоманды.

Макрокоманды выполняются в порядке сверху вниз по тексту макроса. Чтобы переместить команду в макросе вверх или вниз, воспользуйтесь одним из указанных способов:

– перетащите команду в нужное место вверх или вниз;

– выделите команду и нажмите клавиши «**Ctrl + стрелка вверх**» или «**Ctrl + стрелка вниз**»;

– выделите команду, а затем щелкните стрелку **Вверх** или **Вниз** в правой части области макроса.

Удаление макрокоманды

Чтобы удалить макрокоманду, выделите команду и нажмите клавишу **Delete**. Кроме того, можно нажать кнопку **Удалить (X)** в правой части области макроса.

Примечания:

– при удалении блока макрокоманд (например, блока **Если** или **Группа**) удаляются все действия в этом блоке;

– команды **Вверх**, **Вниз** и **Удалить** также доступны в контекстном меню макрокоманды.

Запуск макроса

Запустить макрос можно любым из указанных ниже способов:

– дважды щелкнуть макрос в области навигации;

– вызвать макрос с помощью макрокоманды **ЗапускМакроса** или **ПриОшибке**;

– указать имя макроса в свойстве **Событие** любого объекта. В результате макрос будет выполнен при возникновении этого события.

Обмен данными

MS Access позволяет осуществлять обмен данными с БД MS Access, другими СУБД, программами работы с электронными таблицами (Excel и Lotus), а также импортировать и экспортировать данные из текстовых файлов. Кроме того, с помощью этих средств можно копировать объекты из одной базы данных MS Access в другую.

Экспорт данных позволяет использовать информацию, сохраненную в Access-базе данных при работе с другой программой ми реализуется следующим образом:

1) в области навигации выберите из списка таблицу, которую предполагается экспортировать;

2) откройте контекстное меню и выберите команду **Экспорт**;

3) выберите приложение в которое будет экспортирована таблица;

4) нажмите кнопку **Экспорт;**

5) в открывшемся окне диалога укажите необходимые опции;

6) нажмите **ОК**. Таблица будет сохранена в новом файле указанного типа.

Импорт данных. MS Access может считывать данные, представленные в другом формате, и сохранять их в новой таблице данных, что реализуется следующим образом:

1) перейдите в область навигации и вызовите контекстное меню;

2) выберите команду **Импорт** и приложение из которого будут импортироваться данные;

3) в открывшемся окне диалога укажите необходимые опции.

Задание для самостоятельного выполнения

1. Создайте еще одну БД.

2. Используя импорт данных, поместите в нее таблицы «Товары 1» и «Склады», форму «Товары 1», созданную в результате выполнения шестого задания предыдущего упражнения и запрос на обновление сезонных скидок.

3. Создайте макрос, позволяющий при каждом новом открытии этой БД открывать форму с данными о телевизорах.

4. Откройте БД «Продажи».

5. Используя экспорт данных, поместите таблицы «Фирмы» и «Продажи» в созданную в этом упражнении БД и перейдите в нее.

6. Создайте запрос на отображение данных о названиях товаров, их марке, ценах и данных о складах, где хранятся эти товары.

7. На основе этого запроса создайте простую форму.

8. Создайте макрос, позволяющий в созданной форме выполнять команду фильтрации записей по названиям товаров.

9. Поместите в созданную простую форму кнопку, при нажатии на которую будет выполняться этот макрос.

Контрольные вопросы:

1. Дайте характеристику основным элементам интерфейса MS Access.

2. Перечислите основные действия для создания таблиц базы данных.

3. Опишите процедуру связывания таблиц и обеспечения целостности данных.

4. Какие манипуляции с данными можно производить в режиме формы и таблицы?

5. Какие возможности фильтрации данных поддерживает MS Access?

6. Проанализируйте ключевые этапы создания запроса. Какие виды запросов можно создавать в MS Access?

7. Сущность перекрестного запроса, процедура его создания.

8. Как реализуются вычисления в запросах?

9. Опишите процесс создания форм и отчетов для связанных таблиц.

10. Сущность макроса, его роль в работе современной базы данных.

ИСПОЛЬЗОВАНИЕ СТРУКТУРИРОВАННОГО ЯЗЫКА ЗАПРОСОВ SQL В УПРАВЛЕНИИ ДАННЫМИ БД MS ACCESS

SQL (Structured Query Language) представляет собой непроцедурный язык, используемый для управления данными реляционных СУБД. Термин «непроцедурный» означает, что на данном языке можно сформулировать, что нужно сделать с данными, но нельзя задать конкретный алгоритм, как именно это следует сделать. Иными словами, в этом языке отсутствуют алгоритмические конструкции, такие как присваивания, операторы цикла, разветвления, переключатели и др.

Язык SQL был создан в начале 70-х годов в результате исследовательского проекта IBM, целью которого было создание языка манипуляции реляционными данными. Официальный стандарт SQL был опубликован ANSI (American National Standards Institute – Национальный институт стандартизации, США) в 1986 году. Затем этот стандарт был расширен в 1989 и 1992 годах, поэтому стандарт SQL носит название SQL92, и это наиболее часто используемая версия SQL. В настоящее время опубликован стандарт SQL3, содержащий некоторые объектно-ориентированные расширения.

Существует три уровня соответствия стандарту ANSI – начальный, промежуточный и полный. Многие производители серверных СУБД, такие как IBM, Informix, Microsoft, Oracle и Sybase, применяют собственные реализации SQL, основанные на стандарте ANSI (отвечающие как минимум начальному уровню соответствия стандарту) и содержащие некоторые расширения, специфические для данной СУБД.

Принципы работы SQL

Предположим, что имеется база данных, управляемая с помощью какой-либо СУБД. Для извлечения из нее данных используется запрос, сформулированный на языке SQL. СУБД обрабатывает этот запрос, извлекает

запрашиваемые данные и возвращает их. Результат выполнения запроса – это всегда таблица, содержащая выбранные из базы данные. Этот процесс схематически изображен на рисунке 54.

Необходимо отметить, что помимо извлечения данных, SQL позволяет добавлять, удалять и изменять данные, определять структуру данных, ограничивать или предоставлять доступ к данным, поддерживать ссылочную целостность и многое другое.

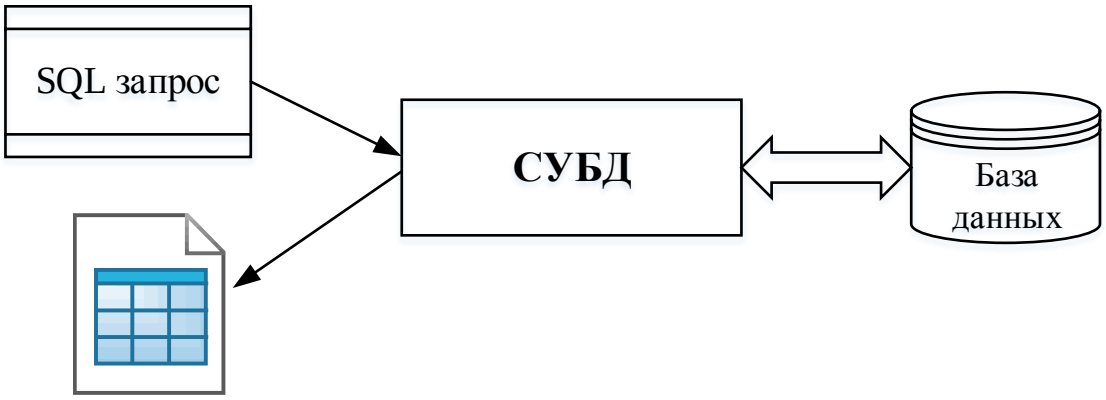


Рис. 54 – Выполнение SQL запроса

При этом, SQL сам по себе не является ни СУБД, ни отдельным продуктом. Это язык, применяемый для взаимодействия с СУБД и являющийся в определенном смысле ее неотъемлемой частью.

Состав операторов языка SQL

SQL содержит примерно 40 операторов для выполнения различных операций над данными, хранящимися в базе данных, с помощью СУБД. Эти операторы подразделяются на пять категорий:

– операторы определения данных (Data Definition Language, DDL);

– операторы манипулирования данными (Data Manipulation Language, DML);

– операторы управления транзакциями (Transaction Control Language, TCL);

– операторы определения доступа к данным (Data Control Language, DCL);

– операторы управления курсорами (Cursor Control Language, CCL).

Рассмотрим функционал каждой категории.

Data Definition Language (DDL) содержит операторы, позволяющие создавать, изменять и удалять базы данных и объекты внутри них (таблицы, представления и др.). Эти операторы представлены в табл. 10.

Таблица 10 – Операторы определения данных

Оператор	Описание
CREATE TABLE	Применяется для добавления новой таблицы к базе данных
DROP TABLE	Применяется для удаления таблицы из базы данных
ALTER TABLE	Применяется для изменения структуры имеющейся таблицы
CREATE VIEW	Применяется для добавления нового представления к базе данных
DROP VIEW	Применяется для удаления представления из базы данных
CREATE INDEX	Применяется для создания индекса для данного поля
DROP INDEX	Применяется для удаления существующего индекса
CREATE SCHEMA	Применяется для создания новой схемы в базе данных
DROP SCHEMA	Применяется для удаления схемы из базы данных
CREATE DOMAIN	Применяется для создания нового домена
ALTER DOMAIN	Применяется для переопределения домена
DROP DOMAIN	Применяется для удаления домена из базы данных

Data Manipulation Language (DML) содержит операторы, позволяющие выбирать, добавлять, удалять и модифицировать данные. Обратите внимание на то, что эти операторы не обязаны завершать транзакцию, внутри которой они вы-званы. Операторы DML перечислены в таблице 11.

Таблица 11 – Операторы манипулирования данными

Оператор	Описание
SELECT	Применяется для выбора данных
INSERT	Применяется для добавления строк к таблице
DELETE	Применяется для удаления строк из таблицы
UPDATE	Применяется для изменения данных

Иногда оператор SELECT относят к отдельной категории, называемой *языком запроса данных* (Data Query Language, DQL).

Операторы *Transaction Control Language (TCL)* применяются для управления транзакциями, т.е. изменениями, выполненными группой операторов DML. Операторы TCL перечислены в таблице 12.

Таблица 12 – Операторы управления транзакциями

Оператор	Описание
COMMIT	Применяется для завершения транзакции и сохранения изменений в базе данных
ROLLBACK	Применяется для отката транзакции и отмены изменений в базе данных
SET TRANSACTION	Применяется для установки параметров доступа к данным в текущей транзакции

Операторы *Data Control Language (DCL),* иногда называемые операторами языка управления доступом (Access Control Language, ACL), применяются для выполнения функций администрирования данных, присваивающих или отменяющих право (привилегию) использовать базу данных, таблицы в базе данных, а также выполнять те или иные операторы SQL. Операторы DCL представлены в таблице 13.

Таблица 13 – Операторы определения доступа к данным

Оператор	Описание
GRANT	Применяется для присвоения прав доступа (привилегии)
REVOKE	Применяется для отмены права доступа (привилегии)

Операторы *Cursor Control Language (CCL)* используются для определения курсора, подготовки SQL-предложений для выполнения, а также для некоторых других операторов. Операторы CCL представлены в таблице 14.

Таблица 14 – Операторы управления курсорами

Оператор	Описание
1	2
DECLARE CURSOR	Применяется для определения курсора для запроса
EXPLAIN	Применяется для описания плана запроса. Этот оператор представляет собой расширение SQL для Microsoft SQL Server 7.0. Он не обязан выполняться в других СУБД.

Продолжение табл. 14

1	2
OPEN CURSOR	Применяется для открытия курсора при получении результатов запроса
FETCH	Применяется для получения строки из результатов запроса
CLOSE CURSOR	Применяется для закрытия курсора
PREPARE	Применяется для подготовки оператора SQL для выполнения
EXECUTE	Применяется для выполнения оператора SQL
DESCRIBE	Применяется для описания подготовленного запроса

Использование операторов языка SQL для выборки данных

С целью закрепления практических навыков использования конструкций языка SQL, построим базу данных «DreamHouse» для некоторой компании, занимающейся сдачей объектов недвижимости. Для корректного отображения предметной области в БД должны быть предусмотрены следующие сущности и их атрибуты:

– *организационная структура компании – филиалы* (таблица *Branch*). О каждом филиале должны храниться такие данные, как номер филиала, адрес (включая город, улицу и номер дома, почтовый индекс), табельный номер менеджера (управляющего) филиала. В каждом филиале есть один и только один менеджер из числа сотрудников этого филиала. Работой всей компании в целом руководит директор, который является сотрудником одного из филиалов. Менеджер филиала руководит работой всего филиала, в котором, кроме него, работают также инспектора и их помощники-ассистенты;

– персонал компании (таблица *Staff*). О каждом сотруднике должны храниться такие данные, как табельный номер, имя (включая имя и фамилию), должность, пол, дата рождения (DateOfBirth) и имя руководителя (если он имеется). Сотрудники компании, занимающие должность инспектора, могут руководить работой нескольких ассистентов (количество которых в любой момент времени не может превышать максимального значения, равного 10).

– *объекты недвижимости*, предназначенные для сдачи в аренду (таблица *PropertyForRent*). О каждом объекте недвижимости, предназначенном для

сдачи в аренду, должны храниться такие данные, как номер объекта недвижимости, адрес (улица, город и почтовый индекс), тип объекта недвижимости, количество комнат, ежемесячная арендная плата и сведения о владельце. Ставка ежемесячной арендной платы для каждого объекта недвижимости пересматривается один раз в год. Основную часть объектов недвижимости, сдаваемых в аренду компанией DreamHouse, составляют квартиры. Однако могут предлагаться в аренду дома, офисы, дачи и другие типы объектов. Управление объектом недвижимости, который сдается в аренду или требуется для аренды, возлагается на одного из сотрудников компании. Любой сотрудник компании может управлять одновременно несколькими объектами недвижимости, количество которых не может превышать 100. Каждый объект недвижимости предлагается в аренду одним из филиалов.

– *владельцы объектов недвижимости* (таблица *PrivateOwner*). Владельцы объектов недвижимости подразделяются на два типа: владельцы частной собственности и владельцы деловых предприятий. О каждом владельце частной собственности хранятся такие данные, как номер владельца, имя (включая имя и фамилию), адрес и номер телефона.

– *клиенты* (таблица *Client*). При регистрации будущего клиента компании DreamHouse в базу данных вносятся такие сведения, как номер клиента, имя (включая имя и фамилию), номер телефона, а также некоторая информация об искомом объекте недвижимости, включая предпочитаемый тип объекта недвижимости и максимальную арендную плату, которую клиент готов платить;

– *осмотр объектов недвижимости* (таблица *Viewing*). Клиент может потребовать, чтобы ему разрешили осмотреть объект недвижимости (в том числе повторно). По результатам каждого осмотра в базу данных вносятся такие сведения, как номер клиента, номер объекта недвижимости, дата осмотра клиентом объекта недвижимости, а также все комментарии, сделанные клиентом по поводу пригодности для него этого объекта недвижимости. Клиент не может осматривать один и тот же объект недвижимости в определенную дату больше одного раза;

– *договора аренды* (таблица *Lease*). После того как клиент находит подходящий для него объект недвижимости, заключается договор аренды. О каждом договоре аренды хранится такая информация, как номер договора аренды, номер клиента, номер объекта недвижимости, ежемесячная арендная плата, метод оплаты, залог, отметка о внесении залога, дата начала и окончания периода аренды, а также продолжительность договора аренды в месяцах. Номер каждого договора аренды является уникальным во всех отделениях компании DreamHouse. Клиент может заключить договор аренды любого объекта недвижимости на срок, не меньше трех месяцев и не превышать одного года. Дата окончания договора аренды остается незаполненной в течение действия договора (договор может быть продлен). Поле заполняется только в момент фактического окончания договора.

Структура данных БД представлена на рисунке 55.

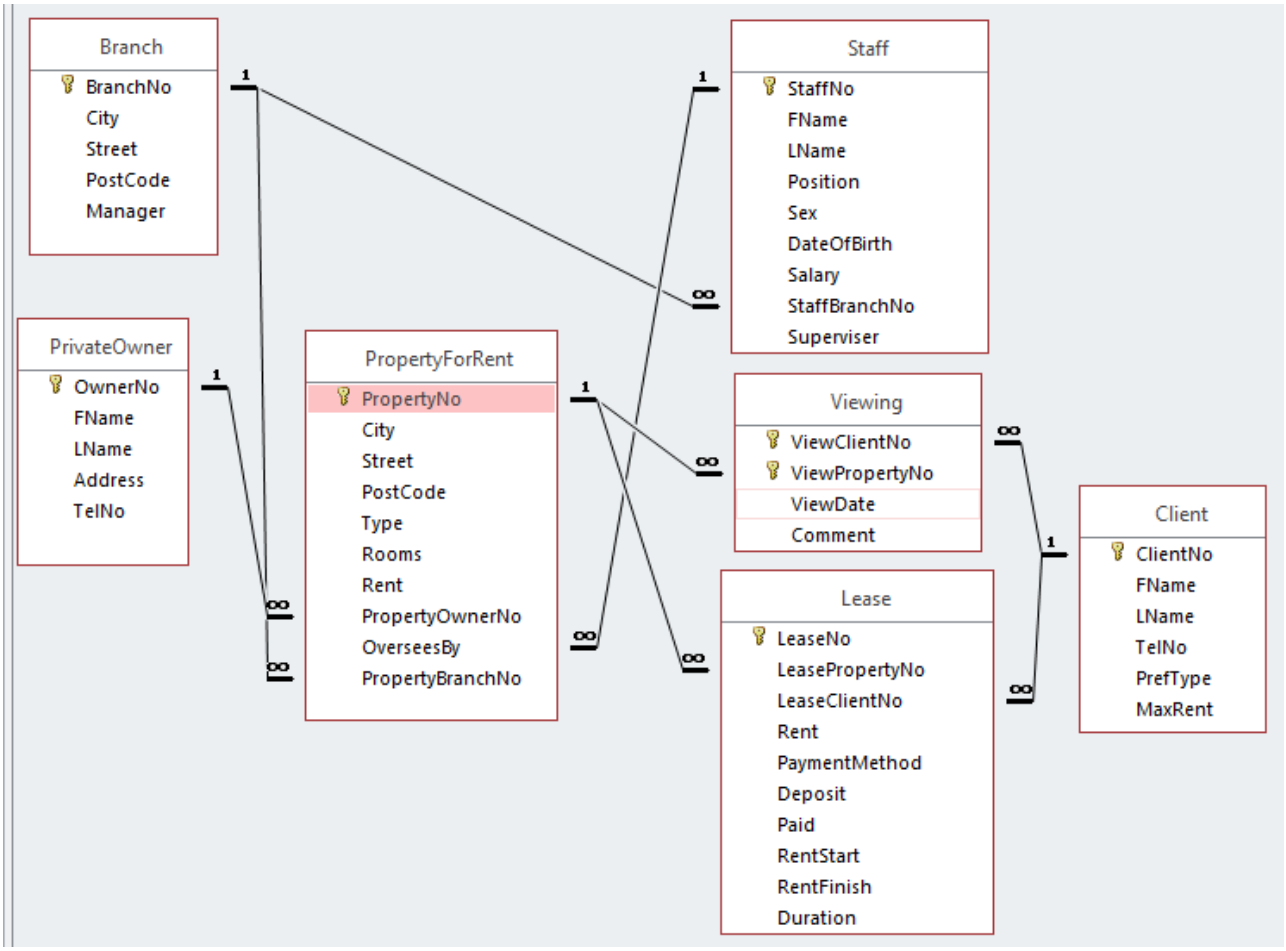


Рис. 55 – Схема данных БД «DreamHouse»

Выборка данных, хранящихся в базе данных, представляет собой наиболее часто встречающуюся операцию, выполняемую с помощью SQL. Оператор **SELECT** один из самых важных операторов этого языка, применяемый для выборки данных.

Синтаксис инструкции **SELECT** имеет следующий вид:

SELECT [ALL | DISTINCT | DISTINCTROW | TOP] { * | table.* | [ta-ble.]fieldl [AS aliasl] [, [table.]field2 [AS alias2] [,. ...]]}

FROM tablel [tablelAlias] [, table2 [table2Alias] [,...]

[WHERE criteria**]**

[GROUP BY groupfieldlist **]**

[HAVING groupcriteria**]**

[ORDER BY fieldl [ASC | DESC][, field2 [ASC | DESC]][, ...]]]

[WITH OWNERACCESS OPTION]

Инструкция SELECT включает следующие основные элементы:

– **SELECT** означает, что с некоторых таблиц базы данных необходимо выбрать набор полей или таблицу данных;

– необязательные слова **ALL**, **DISTINCT**, **DISTINCTROW** и **TOP** называются сущностями (predicates) и определяют выбор следующим образом:

а) **ALL** указывает, что в набор передаются все строки (даже с повторяющимися значениями);

б) **DISTINCT** указывает, что в набор передаются только неповторяющиеся строки;

в) **DISTINCTROW** указывает, что в набор будет включен каждый строку, в которой разница в значении любого поля исходных таблиц;

г) **TOP** используется для отображения некоторого количества (точно или в процентах) начальных или конечных записей с исходного набора;

– список {* | table. * | [Table.] Fieldl [AS alias1], [table.] Field2 [AS alias2] [...]]} (скобки {} здесь обозначают список) состоит из имен полей таблиц запроса. Звездочка «*****» означает выбор всех полей таблицы. Если в запросе

указываются несколько таблиц, то для определения поля используется название таблицы, отделяемое от имени поля точкой «**.**». Поле может получить «алиасное» имя с помощью ключевого слова **AS**.

– после слова **FROM** указываются таблицы, из которых выбираются ранее указанные поля. Здесь «Tableexpression» – это имя таблицы (или таблиц), содержащий данные, «Externaldatabase» – имя базы данных, если не используется текущая база.

Пример 1. Для выбора одного столбца таблицы применяется следующий синтаксис:

SELECT LName

Пример 2. Выборка нескольких колонок имеет вид:

SELECT LName, FName, Position

Пример 3. Если выборка данных осуществляется из нескольких таблиц и при этом выбираются одноименные поля из разных таблиц, надо дополнительно указывать имена таблиц для полной идентификации полей, включаемых в результирующую таблицу данных:

SELECT Staff.LName, Client.LName

Предложение SELECT

В предложении SELECT указывается, какие данные надо получить в результирующей таблице.

Как было сказано ранее, в список выбора могут входить поля таблиц, из которых выполняется выборка. В результирующей таблице элементам списка выбора можно дать новые имена с помощью ключевого слова **AS**.

Пример 4.

SELECT StaffNo AS [Номер сотрудника]

FROM Staff

Над полями таблиц, из которых выполняется выборка, можно выполнять вычисления, а также применять математические и строковые функции, а также агрегатные функции.

***Пример 5*:** Вычисления в предложении SELECT:

SELECT StaffNo AS [Номер сотрудника], Salary/8.30 AS [Оклад в долларах] FROM Staff

Предложение FROM. Для указания имен таблиц, из которых выбираются записи, применяется ключевое слово **FROM**.

Пример 6. Этот запрос возвратит все поля из таблицы Staff.

SELECT *
FROM Staff

Пример 7. Если в результирующей таблице нужны только поля LName и FName, мы можем ввести следующее предложение SELECT:

SELECT LName, FName
FROM Staff

Пример 8. Пример запроса более чем к одной таблице:

SELECT Staff.LName, Client.LName
FROM Staff, Client

Предложение WHERE

Для фильтрации результатов, возвращаемых оператором SELECT, используем предложение WHERE, синтаксис которого имеет вид:

WHERE выражение1 [{AND | OR} выражение2 […]]

Пример 9. При необходимости, вместо получения полного списка сотрудников можно выбрать только тех из них, у которых значение поля Position (должность) равно ‘Менеджер’:

SELECT *

FROM Staff

WHERE Position = ‘Менеджер’

В предложении WHERE можно использовать различные выражения.

Пример10.

SELECT *

FROM Lease

WHERE LeaseClientNo = ‘C0012’ AND Duration > 6

Пример11.

SELECT PropertyNo, City, Street, PostCode

FROM PropertyForRent

WHERE Type = ‘Квартира’ OR Rooms < 3

Пример12.

SELECT LeaseNo, Rent,

FROM Lease

WHERE RentFinish IS NOT NULL

Выражение **IS NOT NULL** означает, что соответствующий столбец результирующей таблицы не может содержать пустые значения.

В предложении WHERE можно использовать один из шести операторов отношений, а также специальные операторы сравнения, приведенные в табл. 15.

Таблица 15 – Операторы отношений, применяемые в выражениях

Оператор	Описание
1	2
<	Меньше
<=	Меньше или равно
<>	Не равно
=	Равно
>	Больше
>=	Больше или равно

Продолжение табл. 15

1	2
ALL	Применяется совместно с оператором сравнения при сравнении со списком значений
ANY	Применяется совместно с операторами сравнения при сравнении со списком значений
BETWEEN	Применяется при проверке нахождения значения внутри заданного интервала (включая его разницы)
IN	Применяется для проверки наличия значения в списке
LIKE	Применяется при проверке соответствия значения заданной маске

Приведем несколько примеров применения этих операторов. Для сопоставления данных с маской применяется ключевое слово **LIKE.**

Пример 13. SELECT LName, FName

FROM Client

WHERE LName LIKE ‘M%’

В данной маске символ «**%**» (процент) заменяет любую последовательность символов, а символ «**_**» (подчеркивание) – один любой символ. Тот же самый результат может быть получен следующим способом.

Пример 14.

SELECT LName, FName

FROM Client

WHERE FName BETWEEN ‘M’ AND ‘H’

Пример 15. Расширим область поиска примера 14. В частности, при поиске Клиентов по фамилиям, начинающимся с букв от А до Д, можно выполнить следующий оператор SELECT:

SELECT LName, FName

FROM Client

WHERE LName BETWEEN ‘А’ AND ‘Д’

Используя оператор **LIKE**, можем сузить диапазон поиска, применив более сложную маску для сравнения.

Пример 16. Чтобы найти клиентов, фамилии которых содержат подстроку ‘ван’, можно применить следующий запрос:

```
SELECT LName, FName
FROM Client
WHERE LName LIKE ‘%ван%’
```

Маска ‘%ван%’ показывает, что до и после искомой подстроки может быть любое количество произвольных символов.

Используя оператор **IN**, можно задать список значений, в котором должно содержаться значение поля.

Пример 17.

```
SELECT LName, FName
FROM Client
WHERE ClientNo IN (‘C0127’, ‘C0315’, ‘C0012’)
```

Операторы AND, OR и NOT

Выше уже был рассмотрен пример применения оператора **AND** для логических операций, связанных с требованием, чтобы запись удовлетворяла двум разным критериям. Проанализируем следующий запрос.

Пример 18. Результатом выполнения этого запроса будет список клиентов, которые хотят арендовать дом, фамилия которых начинается с буквы ‘C’:

```
SELECT LName, FName
FROM Client
WHERE LName LIKE ‘C%’ AND PrefType = ‘дом’
```

Оператор **OR** позволяет выбрать записи, удовлетворяющие хотя бы одному из перечисленных условий, в то время как оператор **NOT** используется для исключения из выборки записей, удовлетворяющих данному условию.

Пример 19. Изменим оператор OR для поиска всех клиентов, которые либо хотят арендовать дом, либо имеющих фамилию, начинающуюся с буквы ‘С’ (и при этом не важно, что они хотят арендовать):

SELECT LName, FName

FROM Client

WHERE LName LIKE ‘С%’ OR PrefType = ‘дом’

В этом случае результирующая таблица будет содержать записи, в которых значение поля LName удовлетворяет первому условию, плюс все записи, в которых значение поля PrefType удовлетворяет второму условию.

Теперь рассмотрим пример применения оператора **NOT**.

Пример 20. Для исключения некоторых клиентов из результирующей таблицы можно использовать запрос вида:

SELECT LName, FName

FROM Client

WHERE PrefType NOT IN (‘дача’, ‘офис’)

В результате выполнения этого запроса получим список клиентов, которые хотят арендовать любой объект недвижимости, кроме дачи и офиса.

Предложение ORDER BY

Предложение **ORDER BY** (необязательное) применяется для сортировки строк результирующей таблицы по одному или нескольким столбцам. Для определения порядка сортировки используются ключевые слова **ASC** (по возрастанию) или **DESC** (по убыванию). По умолчанию данные сортируются по возрастанию. Синтаксис предложения **ORDER BY** имеет вид:

ORDER BY столбец1 [{ASC | DESC}][, столбец2 [{ASC | DESC}] [,…]

Пример 21. Для сортировки сотрудников по фамилии и затем по имени следует использовать следующий SQL-запрос:

```
SELECT LName, FName, Position
FROM Staff
ORDER BY LName, FName
```

Если сортировка данных требуется в убывающем порядке (например, необходимо получить список объектов недвижимости в порядке убывания арендной платы), используется ключевое слово **DESC**.

Пример 22.

```
SELECT PropertyNo, City, Street, PostCode, Rent
FROM PropertyForRent
ORDER BY Rent DESC
```

Связывание таблиц

Как было уже рассмотрено, можно создавать запросы, позволяющие извлечь данные из нескольких таблиц. Одна из возможностей сделать это заключается в связывании таблиц по одному или нескольким полям. Отметим, что без связывания таблиц в результате запроса получится таблица, содержащая все возможные комбинации строк каждой из исходных таблиц (эта операция известна также как декартово произведение).

Пример 23.

```
SELECT Staff.*, Branch.*
FROM Staff, Branch
```

В то время как запрос, показанный ниже, приводит к отображению списка сотрудников с указанием, в каком филиале он работает.

Пример 24.

```
SELECT Staff.*, Branch.*
```

FROM Staff, Branch

WHERE Staff.StaffBranchNo = Branch.BranchNo

В общем случае синтаксис для связывания таблиц имеет вид:

SELECT список_полей

FROM таблица1, таблица2

WHERE таблица1.столбец1 = таблица2.столбец2

Следующие несколько примеров связывания таблиц характерны для Microsoft Access и Microsoft SQL Server и могут не работать с другими СУБД. Существует несколько типов связывания таблиц.

Пример 25. Следующий оператор SQL осуществляет так называемое внутреннее соединение таблиц (**INNER JOIN**) – в этом случае в результирующей таблице содержатся записи, в которых значения в связанных полях совпадают:

SELECT Staff.*, Branch.*

FROM Staff INNER JOIN Branch ON Staff.StaffBranchNo = Branch.BranchNo

Так называемые внешние соединения (**OUTER JOINS**) позволяют нам включить в результат запроса все строки из одной таблицы и соответствующие им строки из другой таблицы. Если в другой таблице нет строк, соответствующих какой-то строке, то в результат включается пустая строка, т.е. строка, состоящая их всех пустых полей.

Пример 26.

SELECT Client.*, Lease.*

FROM Client LEFT OUTER JOIN Lease ON Client.ClientNo = Lease.LeaseClientNo

Это так называемое левое внешнее соединение (**LEFT OUTER JOIN**). В результирующую таблицу попадут все клиенты, каждая строка будет соединена с договором этого клиента, а в тех случаях, когда у клиента нет ни одного договора – с пустой строкой.

Существуют также правые внешние соединения (**RIGHT OUTER JOIN**), возвращающие все строки из второй (то есть правой) таблицы и соответствующие им строки из другой таблицы.

Пример 27.

```
SELECT Lease.*, PropertyForRent.*
FROM Lease RIGHT OUTER JOIN PropertyForRent ON Lease.LeasePropertyNo = PropertyForRent.PropertyNo
```

В результирующую таблицу попадут все объекты недвижимости, каждая строка будет соединена с договором аренды этого объекта, а в тех случаях, когда на объект договора аренды не заключались – с пустой строкой.

Комбинируя левое и правое внешние соединения, можно получить полное внешнее соединение, возвращающее все данные из обеих таблиц.

Пример 28.

```
SELECT Client.*, PropertyForRent.*
FROM Client FULL OUTER JOIN PropertyForRent ON Client.PrefType = PropertyForRent.Type
```

Для получения всех комбинаций строк из обеих таблиц (декартова произведения) можно использовать ключевое слово **CROSS JOIN** без указания связываемых полей.

Пример 29.

```
SELECT Staff.*, Branch.*
FROM Staff CROSS JOIN Branch
```

Результат будет этого запроса совпадать с результатом первого запроса этого раздела.

Если в запросе используется более трех таблиц, можно использовать вложенные соединения.

Предложение GROUP BY

Для вычисления суммарных значений на основе данных одной или нескольких таблиц можно использовать предложение **GROUP BY**, имеющее такой синтаксис:

GROUP BY столбец1 [, …]

Пример 30. Запрос связывает две таблицы, сортирует их по полю BranchNo, для каждого значения BranchNo создает одну строку в результирующей таблице и вычисляет количество значений поля StaffBranchNo для каждого значения BranchNo:

SELECT Branch.BranchNo, COUNT (Staff.StaffBranchNo)

FROM Branch INNER JOIN Staff ON Branch.BranchNo = Staff.StaffBranchNo

GROUP BY Branch.BranchNo

В приведенном выше примере запроса использовали в предложении **SELECT** агрегатную функцию **COUNT**, вычисляющую количество значений. В таблице 16 указан список наиболее часто используемых агрегатных функций.

Таблица 16 – Перечень некоторых агрегатных функций языка SQL

Функция	Назначение
1	2
AVG	Вычисляет среднее арифметическое
COUNT	Вычисляет количество непустых значений в столбце
MAX	Вычисляет наибольшее значение в столбце
MIN	Вычисляет наименьшее значение в столбце
SUM	Вычисляет сумму значений в столбце

Продолжение табл. 16

1	2
STDEV	Вычисляет несмещенное стандартное отклонение для значений в столбце. Эта функция используется только в Microsoft Access и Microsoft SQL Server
STDEVP	Вычисляет смещенное стандартное отклонение для значений в столбце. Эта функция используется только в Microsoft Access и Microsoft SQL Server
VAR	Вычисляет несмещенную дисперсию отклонения для значений в столбце. Эта функция используется только в Microsoft Access и Microsoft SQL Server
VARP	Вычисляет смещенную дисперсию отклонения для значений в столбце. Эта функция используется только в Microsoft Access и Microsoft SQL Server

Помимо перечисленных выше агрегатных функций можно использовать также математические и строковые функции, приведенные в таблице 17.

Таблица 17 – Математические и строковые функции

Функция	Назначение
1	2
ABS	Возвращает абсолютное значение числа
CEIL	Округляет дробное число
FLOOR	Удаляет дробную часть числа
GREATEST	Возвращает наибольшее из двух значений. Эта функция используется только в MS Access и MS SQL Server
LEAST	Возвращает наименьшее из двух значений. Эта функция используется только в MS Access и MS SQL Server
MOD	Возвращает остаток от деления одного числа на другое
POWER	Возвращает значение, равное одному числу в степени, равной другому числу
ROUND	Округляет число с точностью до указанного десятичного знака
SIGN	Возвращает –1, если число отрицательное, и 1, если положительное
SQRT	Вычисляет квадратный корень числа
LEFT	Возвращает указанное число знаков строки, начиная слева. Эта функция используется только в MS Access и MS SQL Server
RIGHT	Возвращает указанное число знаков строки, начиная справа. Эта функция используется только в MS Access и MS SQL Server
UPPER	Заменяет все буквы в строке на прописные
LOWER	Заменяет все буквы в строке на строчные
INITCAP	Расставляет заглавные буквы в начале слов в строке
LENGTH	Вычисляет число символов в строке

Продолжение табл. 17

1	2
LPAD	Добавляет указанный символ в левую часть строки в количестве, необходимом для того, чтобы строка имела заданную длину
RPAD	Добавляет указанный символ в правую часть строки в количестве, необходимом для того, чтобы строка имела заданную длину
SUBSTR	Извлекает подстроку нужной длины из строки, начиная с указанной позиции

Предложение HAVING

Предложение **HAVING** имеет назначение, сходное с предложением WHERE, но используется с агрегатными данными.

Пример 31.

```
SELECT Branch.BranchNo, COUNT (Staff.StaffBranchNo)
FROM Branch INNER JOIN Staff ON Branch.BranchNo = Staff.StaffBranchNo
GROUP BY Branch.BranchNo
HAVING COUNT(Staff.StaffNo) >= 10
```

Этот запрос аналогичен предыдущему, но в результирующую таблицу включены только филиалы, в которых работают десять или более сотрудников.

Ключевые слова ALL и DISTINCT

До этого момента были рассмотрены инструкции, позволяющие извлечь все или заданные столбцы из одной или нескольких таблиц. Для управления выводом дублирующихся строк в результирующей таблице можно использовать ключевые слова **ALL** или **DISTINCT** в предложении **SELECT.** Ключевое слово **DISTINCT** указывает, что строки в результирующей таблицы должны быть уникальны, тогда как ключевое слово **ALL** указывает, что возвращать следует все строки.

Пример 32. Для извлечения названий городов, в которых имеются филиалы фирмы, можно использовать следующий запрос:

```
SELECT DISTINCT City
FROM Branch
```

Отметим, что ключевое слово **ALL** используется по определению. Если в запросе требуется вывести более одного столбца и при этом использовано слово DISTINCT, то результирующая таблица будет содержать различные строки, но некоторые значения одного и того же поля в разных строках могут совпадать.

Ключевое слово TOP

Ключевое слово TOP может быть использовано для возврата первых n строк или первых n процентов таблицы.

Пример 33. Запрос возвращает первые 10 договоров из таблицы:

```
SELECT TOP 10 *
FROM Lease
ORDER BY LeaseNo
```

Пример 34. Запрос возвращает первую четверть записей таблицы:

```
SELECT TOP 25 PERCENT *
FROM Lease
ORDER BY LeaseNo
```

Операторы модификации данных

Помимо извлечения данных, язык SQL может быть использован для обновления и удаления данных, копирования записей в другие таблицы и выполнения многих других операций. Ниже мы рассмотрим операторы **UPDATE, DELETE и INSERT**, используемые для решения некоторых из этих задач.

Оператор UPDATE

Для изменения значений в одной или нескольких колонках таблицы применяется оператор **UPDATE.** Синтаксис этого оператора имеет вид:

UPDATE таблица SET столбец1 = выражение1 [, столбец2 = выражение2] [,…] [WHERE условие_отбора]

Выражение в предложении **SET** может быть константой или результатом вычислений.

Пример 35. Для повышения зарплаты на 10% всем сотрудникам, получающим меньше 10000 руб., можно выполнить следующий запрос:

```
UPDATE Staff
SET Salary = Salary * 1.1
WHERE Salary < 10000
```

Оператор DELETE

Для удаления строк из таблиц следует использовать оператор **DELETE**, синтаксис которого имеет вид:

DELETE
FROM таблица
[WHERE условие_отбора]

При этом, предложение WHERE не является обязательным, но если его не включить, то из таблицы будут удалены все записи.

Пример 36. Для удаления из таблицы всех завершенных договоров, можно выполнить следующий запрос:

```
DELETE
FROM Lease
WHERE RentFinish IS NOT NULL
```

Отметим, что полезно использовать оператор SELECT с тем же синтаксисом, что и оператор DELETE, чтобы проверить, какие именно записи будут удалены, прежде чем действительно их удалять. Ниже показан оператор SELECT для приведенного выше запроса на удаление данных.

Пример 37.

```
SELECT *
FROM Lease
WHERE RentFinish IS NOT NULL
```

Можно использовать в предложении WHERE более сложный критерий для отбора тех записей, которые должны быть удалены.

Пример 38. Предположим, нам нужно удалить из списка клиентов тех из них, кто не имеет действующий договор. Сначала следует выполнить следующий SELECT, чтобы определить, что именно удаляем:

```
SELECT Client.*
FROM Client
WHERE ClientNo NOT IN
(SELECT LeaseClientNo
FROM Lease
WHERE RentFinish IS NULL)
```

а затем заменить оператор SELECT на оператор DELETE:

```
DELETE
FROM Client
WHERE ClientNo NOT IN
(SELECT LeaseClientNo
FROM Lease
WHERE RentFinish IS NULL)
```

Оператор INSERT

Для добавления записей в таблицы следует использовать оператор **INSERT**, синтаксис которого имеет вид:

INSERT [INTO] таблица
([список_столбцов]
{ VALUES ({ DEFAULT | NULL | выражение } } [, …]
)

Пример 39. Для добавления нового клиента в таблицу Client можно использовать следующий запрос:

INSERT INTO Client (ClientNo, LNmae, FName, TelNo, PrefType, MaxRent)
VALUES (‘C0098’, ‘Петренко’, ‘Семен’, 7171247, ‘квартира’, 800)

Модификация метаданных в SQL

Существует несколько операторов SQL для управления метаданными, используемых для создания, изменения или удаления баз данных и содержащихся в них объектов (таблиц, представлений и др.). Рассмотрим основные из них: **CREATE TABLE, ALTER TABLE и DROP**.

Оператор CREATE TABLE

Для создания новой таблицы необходимо использовать оператор **CREATE TABLE**, синтаксис которого имеет вид:

CREATE TABLE таблица
(столбец1 тип1 [(длина1)][CONSTRAINT ограничение_на_столбец1]
[, столбец2 тип2 [(длина2)][CONSTRAINT ограничение_на_столбец2] [, ...]]
[CONSTRAINT ограничение_на_таблицу1 [, ограничение_на_таблицу2 [, ...]]])

В этом операторе следует указать имя поля, тип данных для него (тип данных должен поддерживаться данной СУБД), длину (для некоторых типов полей) и, если нужно, серверные ограничения (с применением ключевого слова **CONSTRAINT**).

Пример 40. Представленный запрос создает таблицу с именем Simple с четырьмя колонками – LastName, FirstName, EMail и HomePage:

```
CREATE TABLE Simple
( FirstName varchar(50) NOT NULL,
  LastName varchar(50) NOT NULL,
  EMail varchar(50),
  HomePage varchar(255)
)
```

Пример 41. Можем расширить эту таблицу добавлением поля PersonID, которое будет использовано как первичный ключ:

```
CREATE TABLE Simple
( PersonID Integer NOT NULL PRIMARY KEY,
  FirstName varchar(50) NOT NULL,
  LastName varchar(50) NOT NULL,
  EMail varchar(50),
  HomePage varchar(255)
)
```

Пример 42. Можем указать, что комбинация полей LastName и FirstName должна быть уникальна:

```
CREATE TABLE Simple
( PersonID Integer NOT NULL PRIMARY KEY,
  FirstName varchar(50) NOT NULL,
  LastName varchar(50) NOT NULL,
  EMail varchar(50),
```

HomePage varchar(255),

CONSTRAINT SimpleConstraint UNIQUE (FirstName, LastName)

)

Используя предложение **SELECT** и ключевое слово **INTO**, можем заполнять новые таблицы значениями из существующих таблиц, отбирая для них данные по условию, указанному в предложении **WHERE**.

Пример 43.

```
SELECT *
INTO NewLease
FROM Lease
WHERE RentStart > Date('01/01/19') AND RentFinish IS NULL
```

Этот запрос создаст новую таблицу NewLease и заполнит ее данными о действующих договорах аренды, с 1 января 2019 года.

При использовании в операторах SQL даты или времени, а также полей, содержащих такие данные, следует уточнить синтаксис таких предложений в документации из комплекта поставки используемой СУБД.

Оператор ALTER TABLE

Для изменения структуры существующей таблицы можно использовать оператор **ALTER TABLE**. Применяя его, можно добавить или удалить поле или серверное ограничение. Существует четыре разновидности оператора ALTER TABLE.

Первая разновидность этого оператора используется для добавления колонки к таблице, и ее синтаксис имеет вид:

ALTER TABLE таблица ADD [COLUMN] столбец тип_данных [(длина)] [CONSTRAINT ограничение_на_столбец]

В запросах такого вида определяется имя таблицы, имя нового поля, его тип данных и, если нужно, размер. Помимо этого, можно указать серверное ограничение, связанное с данным полем.

Пример 44. Для добавления поля Phone к таблице Simple, созданной ранее, можно выполнить следующий запрос:

ALTER TABLE Simple ADD Phone varchar(30)

Вторая разновидность оператора ALTER TABLE применяется для добавления серверных ограничений к таблице, а ее синтаксис имеет вид:

ALTER TABLE таблица ADD CONSTRAINT ограничениие_на_таблицу

Такие запросы позволяют только добавлять индексы, позволяющие использовать соответствующие поля в качестве первичных или внешних ключей.

Третья разновидность предложения ALTER TABLE применяется для удаления поля из таблицы, при этом ключевое слово **COLUMN** использовать не обязательно:

ALTER TABLE таблица DROP [COLUMN] столбец

Пример 45.

ALTER TABLE Simple DROP Phone

Для удаления проиндексированных полей следует сначала удалить индекс. Это можно сделать с помощью *четвертой разновидности предложения* ALTER TABLE:

ALTER TABLE таблица DROP CONSTRAINT индекс

Пример 46.

ALTER TABLE Simple DROP CONSTRAINT PrimaryKey

Оператор DROP

Для удаления таблиц или индексов можно использовать оператор DROP, имеющий две разновидности.

Первая разновидность применяется для удаления таблицы из базы данных:

DROP TABLE таблица

Вторая разновидность используется для удаления индекса:

DROP INDEX индекс ON таблица

Контрольные вопросы:

1. Объясните назначение следующих предложений, входящих в инструкцию SELECT: ORDER BY; GROUP BY; WHERE.

2. Что позволяет сделать ключевое слово DISTINCT?

3. В чем отличие между предложениями WHERE и HAVING?

4. Для чего применяются агрегирующие функции?

5. Каким образом в запросе можно обработать значения NULL?

6. Каким образом можно соединить 2 (и более) таблицы в SQL?

7. Раскройте порядок применения инструкции INSERT.

8. Почему не следует применять инструкции UPDATE и DELETE без предложения WHERE?

9. Какие возможности по модификации таблиц предоставляет DDL?

10. Каким образом при создании таблиц описываются правила обеспечения ссылочной целостности?

СПИСОК ЛИТЕРАТУРЫ

1. Введение в системы баз данных: учебное пособие / П. В. Бураков, В. Ю. Петров. – СПб: НИУ ИТМО, 2010. – 126 с.

2. Диго, С. М. Базы данных проектирование и использование: учебник / С. М. Диго. – М.: Финансы и статистика, 2005. – 592 с.

3. Рикарди, Г. Системы баз данных. Теория и практика использования в Internet и среде Java / Г. Рикарди. – М.: Издательский дом «Вильямс», 2001. – 480 с.

4. Илюшечкин, В. М. Основы использования и проектирования баз данных: учебник для СПО / В. М. Илюшечкин. – М.: Издательство Юрайт, 2019. – 213 с.

5. Осипов, Д. Л. Технологии проектирования баз данных / Д. Л. Осипов. – М.: ДМК Пресс, 2019. – 498 с.

6. Коннолли, Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли. К. Бегг, А. Страчан.; пер. с англ. – 2-е изд. – М.: Вильямс, 2001. – 1120 с.

7. Роб, П. Системы баз данных: проектирование, реализация и управление / П. Роб, К. Коронел.; пер с англ. – 5-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2004. – 1040 с.

8. Теория и практика построения баз данных / Д. Крёнке. – СПб.: Питер, 2003. – 800 с.

9. ГОСТ 34.321–96. Информационные технологии. Система стандартов по базам данных. Эталонная модель управления данными.

10. Codd, E. F. (1982) The 1981 ACM Turing Award Lecture: Relational database: A practical foundation for productivity. Comm. ACM, 25(2), 109–117.

11. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс; пер. с англ. М.: Русская редакция, 2004. – 576 с.

12. Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004 – 655 с.

13. Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул; под ред. Л. Г. Гагариной. – М.: ИД «ФОРУМ»: ИНФРА-М, 2008. – 400 с.

14. Иванова, Г. С. Технология программирования: учебник / Г. С. Иванова. – М.: КНО-РУС, 2011. – 336 с.

15. Тамре, Л. Введение в тестирование программного обеспечения / Л. Тамре; пер. с англ. – М.: Вильямс, 2003. – 368 с.

16.Райордан, Р. Основы реляционных баз данных / Р. Райордан ; пер. с англ. М.: Русская редакция, 2001. – 384 с.

17. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буг ; пер. с англ. М.: Вильямс, 2010. – 720 с.

18. Стоунбрейкер, М. Объектно-реляционные системы баз данных / М. Стоунбрейкер // Открытые системы. – 1994. – № 4.

19. Гарольд, Э. XML: справочник / Э. Гарольд, С. Минс; пер. с англ. – СПб.: Символ-Плюс, 2002. – 576 с.

20. Хантер, Д. XML. Базовый курс / Д. Хантер, Дж. Рафтер, Д. Фаусетт, Э. ван дер Влист и др.; пер. с англ. – 4-е изд. – М.: ООО «И.Д. Вильямс», 2009. – 1344 с.

21. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. – 8-е изд. ; пер. с англ. М.: Вильямс, 2006. – 1328 с.

22. Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL / К. Дж. Дейт. ; пер. с англ. СПб.: Символ-Плюс, 2010. – 480 с.

23. Гарсия-Молина, Гектор Системы баз данных. Полный курс / Гарсия-Молина Гектор, Ульман Джеффри Д, Уидом Дженнифер; пер. с англ. М.: Вильямс, 2003. – 1088 с.

24. Тарасов, В. Л. Работа с базами данных в Access 2010. Часть 1: учебно-методическое пособие / В. Л. Тарасов. – Нижний Новгород: Нижегородский госуниверситет, 2014. – 126 с.

25. Одиночкина, С. В. Разработка баз данных в Access 2010 / С.В. Одиночкина. – СПб: НИУ ИТМО, 2012. – 83с.

26. Гурвиц, Г. А. Microsoft Access 2010. Разработка приложений на реальном примере / Г. А. Гурвиц. – СПб.: БХВ-Петербург, 2010. – 496 с.

27. Федоров, Э. Г. Управление данными: курс лекций : учебное пособие для студентов, обучающихся по направлению «Информационные системы» / Э.Г. Федоров, О.В. Коновалов ; Волгогр. гос. архит.-строит. ун-т. – Волгоград : ВолгГАСУ, 2007. – 144 с.

28. Щелоков, С. А. Базы данных: учебное пособие / С. А. Щелоков. – Оренбург: ОГУ, 2014. – 298 с.

29. Грофф, Дж. SQL: Полное руководство / Дж. Грофф, П. Вайнберг; пер. с англ. 2-е изд., перераб. и доп. К.: Издательская группа BHV, 2001. – 816 с.

30. Молинаро, Э. SQL: Сборник рецептов / Э. Молинаро; пер. с англ. СПб.: Символ-Плюс, 2009. – 672 с.

31. Распределенные базы данных : учеб. пособ. / Н. Ю. Братченко. – Ставрополь : СКФУ, 2015. – 130 с.

32. Кошелев, В. Е. Access 2007. Эффективное использование / В. Е. Кошелев. – М.: Бином-Пресс., 2008. – 592 с.

33. Карпова, Т. С. Базы данных: модели, разработка, реализация / Т. С. Карпова. – Питер, 2002. – 304 с.

34. Харрингтон, Д. Л. Проектирование реляционных баз данных / Д. Л. Харрингтон; пер. с англ. И. Дранишников. – М.: Лори, 2006. – 230 с.

35. Бэнкер, К. MongoDB в действии / К. Бэнкер; пер. с англ. А. А. Слинкина. М.:ДМК Пресс, 2012. – 394 с:

36. Свиридова, М. Ю. Система управления базами данных Access / М. Ю. СвиридоваМ.Ю. Свиридова. – М.: Академия, 2016. – 192 с.

37. Adam Shook, Donald Miner. MapReduce Design Patterns. O’Reilly Media, Inc. 2012. – 256 p.

38. Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), p. 137–150.

39. Браст, Э. Дж. Разработка приложений на основе Microsoft SQL Server 2005. Мастер-класс / Э. Дж. Браст, С. Форте; пер. с англ. М.: Русская редакция, 2007. – 880 с.

40. Шаньгин, В. Ф. Защита компьютерной информации. Эффективные методы и средства / В. Ф. Шаныгин. М.: ДМК Пресс, 2008. – 544 с.

41. Редько, В. Н. Базы данных и информационные системы / В. Н. Редько, И. А. Бассараб. – М.: Знание, 2013. – 146 с.

42. Системы управления базами данных для ЕС ЭВМ / ред. В.М. Савинков. – М.: Финансы и статистика, 2013. – 224 с.

43. Тимошок, Б. Самоучитель Microsoft Access 2002 / Б. Тимошок. – М.: Вильямс, 2017. – 352 с.

44. Туманов, В. Е. Основы проектирования реляционных баз данных / В. Е. Туманов. – М.: Бином, 2018. – 420 с.

45. Уэлдон Дж.-Л. Администрирование баз данных / Уэлдон Дж.-Л. – М.: Финансы и статистика, 2014. – 207 с.

Гуменюк Наталья Владимировна

БАЗЫ ДАННЫХ

Учебное пособие
для обучающихся образовательных учреждений
высшего профессионального образования
(на русском языке)

Ответственная за выпуск Н. Ф. Курган
Техническое редактирование и корректура Т. В. Чубучной

Подписано к печати 02.03.2020 г.
Формат 70×90/16. Бумага офисная.
Гарнитура «Times New». Печать – лазерная.
Усл. печ. листов 17,67.
Тираж 100 экз. Заказ № 49.

ГОУВПО «ДОННТУ»
83001, ДНР, г. Донецк, ул. Артема, 58. Тел.: (062)301-03-04

Отпечатано в Автомобильно-дорожном институте ГОУВПО «ДОННТУ»
84646, г. Горловка, ул. Кирова, 51

Свидетельство о государственной регистрации ДНР
Серия АА03 № 029192 от 7 апреля 2016 г.