

Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ
И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

ФАКУЛЬТЕТ ДИСТАНЦИОННОГО ОБУЧЕНИЯ (ФДО)

Д. П. Вагнер

БАЗЫ ДАННЫХ

Учебное пособие

Томск
2017

УДК 004.65(075.8)
ББК 32.973.233-018.2я73
В 125

Рецензенты:

С. И. Колесникова, д-р техн. наук, профессор кафедры высшей математики
и математического моделирования Национального исследовательского
Томского государственного университета;

М. А. Афонасова, д-р экон. наук, профессор, зав. кафедрой менеджмента
Томского государственного университета систем управления
и радиоэлектроники

Вагнер Д. П.

В 125 Базы данных : учебное пособие / Д. П. Вагнер. – Томск : ФДО,
ТУСУР, 2017. – 133 с.

В учебном пособии представлены основы теории и практики использования баз данных. Приведены основные понятия реляционных моделей данных, а также синтаксис и примеры использования языка SQL. Особое внимание уделено описанию процесса проектирования на основе использования принципов нормализации и применения ER-моделей.

© Вагнер Д. П., 2017

© Оформление.

ФДО, ТУСУР, 2017

Оглавление

Введение	5
1 Введение в теорию баз данных	7
1.1 Данные и информация	7
1.2 Базы данных и СУБД	8
1.3 Классификация СУБД	12
1.4 Представление данных в БД	14
1.5 Модели данных	17
1.5.1 Иерархическая модель	17
1.5.2 Сетевая модель	19
1.5.3 Реляционная модель	21
1.5.4 Объектно-ориентированная модель	22
1.5.5 Многомерная модель	23
1.5.6 Постреляционная модель	25
2 Реляционная модель данных	26
2.1 Определения модели	26
2.2 Ключевые атрибуты	29
2.3 Связи в БД	33
2.3.1 Связь «один-к-одному»	33
2.3.2 Связь «один-ко-многим»	35
2.3.3 Связь «многие-ко-многим»	37
2.3.4 Управление связанными записями в БД	38
2.4 Реляционные СУБД	40
2.4.1 Microsoft Access	40
2.4.2 MySQL	46
3 Язык SQL	50
3.1 История SQL	50
3.2 Основные операторы языка SQL	51
3.3 Типы данных	52
3.4 Оператор SELECT	54
3.5 Операторы DML	59
3.5.1 INSERT	59
3.5.2 UPDATE	59
3.5.3 DELETE	60
3.6 Групповые функции	61
3.7 Запросы к нескольким таблицам	64

3.7.1 Декартово произведение записей таблиц.....	65
3.7.2 Соединение таблиц	69
3.7.3 Объединение таблиц.....	72
4 Проектирование баз данных	73
4.1 Жизненный цикл БД.....	73
4.2 Нормализация БД.....	78
4.2.1 Дублирование данных	79
4.2.2 Аномалии.....	81
4.2.3 Теория нормализации	82
4.2.4 Нормальные формы	85
4.3 Инфологическое проектирование БД	92
4.3.1 Модель «сущность – связь»	93
4.3.2 Виды связей между сущностями.....	96
4.3.3 Методология проектирования на основе ER-моделей.....	102
4.3.4 CASE-средства	108
5 Администрирование баз данных.....	113
5.1 Безопасность БД.....	114
5.2 Резервное копирование БД	117
5.3 Настройка доступа к БД.....	119
5.4 Дополнительные инструменты защиты БД.....	121
Заключение.....	124
Литература.....	125
Список условных обозначений и сокращений	127
Глоссарий.....	129
Приложение А SQL-код для создания базы данных.....	132

Введение

В настоящее время информация является одним из ключевых факторов эффективной деятельности во всех сферах жизни человека. С каждым годом мы получаем всё большие объёмы информации, соответственно, повышаются требования к таким аспектам, как поиск, анализ, обработка и скорость передачи информации. Функционирование любой организации немыслимо без осуществления процессов хранения и обработки информации, чаще всего реализованных с помощью инструментария баз данных и информационных систем. Несомненно, в современном мире всё более актуальным становится знание основ функционирования и проектирования баз данных, а также применения современных технологий и программных продуктов систем управления базами данных.

История развития баз данных берет своё начало с конца 1960-х гг., когда была введена в эксплуатацию первая промышленная система управления базами данных (СУБД) IMS фирмы IBM. На самом деле направление использования вычислительной техники, связанное с хранением данных в виде баз данных на первых этапах развития несколько отставало от сферы непосредственной обработки данных. Это было связано прежде всего с ограниченными аппаратными возможностями запоминающих устройств того времени. В дальнейшем развитие аппаратных возможностей устройств, а также разработка реляционной модели данных известным американским математиком Э. Ф. Коддом привели к существенному развитию и последующему широкому распространению систем управления базами данных практически во всех сферах деятельности.

Пособие «Базы данных» включает 5 глав.

В первой главе определяются основные понятия теории баз данных, даются функции и классификация СУБД, а также описание моделей данных.

Вторая глава посвящена описанию реляционной модели данных.

В третьей главе приведено описание синтаксиса основных команд языка SQL.

Четвертая глава посвящена описанию процесса проектирования баз данных на основе принципов нормализации и построения ER-диаграмм.

В пятой главе приведено описание проблем, связанных с администрированием и защитой баз данных.

Соглашения, принятые в учебном пособии

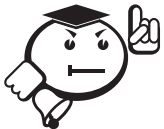
Для улучшения восприятия материала в данном учебном пособии используются пиктограммы и специальное выделение важной информации.



Эта пиктограмма означает определение или новое понятие.



Эта пиктограмма означает «Внимание!». Здесь выделена важная информация, требующая акцента на ней. Автор может поделиться с читателем опытом, чтобы помочь избежать некоторых ошибок.



В блоке «На заметку» автор может указать дополнительные сведения или другой взгляд на изучаемый предмет, чтобы помочь читателю лучше понять основные идеи.



Пример

Эта пиктограмма означает пример. В данном блоке автор может привести практический пример для пояснения и разбора основных моментов, отраженных в теоретическом материале.



Контрольные вопросы по главе

1 Введение в теорию баз данных

1.1 Данные и информация

Во все времена решение большинства задач было непосредственно связано с получением и анализом необходимой для этого информации. Однако сам термин «информация» использовался не столь часто. Ещё позже появился термин «данные», который во многих случаях применялся как синоним информации, т. е. термин близкий к ней по смыслу. Сегодня термины «информация» и «данные» употребляются в повседневной жизни практически повсеместно, однако различия между этими понятиями для многих являются неочевидными.

Из теории информатики известны и наиболее часто употребляются следующие определения данных:

- данные – это зарегистрированные сигналы, которые несут в себе сведения об окружающем нас мире;
- данные – *информация*, подлежащая вводу в ЭВМ;
- данные – это *информация* о событиях, произошедших в материальном мире, представленная в формализованном виде, пригодном для хранения, передачи и обработки.

Как видно, данные не совсем тождественны информации, хотя эти термины зачастую встречаются в определениях друг друга.

Для понимания различий рассмотрим пример: «наблюдая излучения далеких звезд, человек получает определенный поток данных, но станут ли эти данные информацией, зависит от целого ряда других обстоятельств» [1].

В дальнейшем в ходе изучения предмета «Базы данных» для однозначной интерпретации терминов вводятся следующие определения:



.....
Данные – это сведения об объектах окружающего мира, введенные на специальный носитель и предназначенные для хранения, передачи и обработки.

Информация – это совокупность сведений о фактических данных и зависимостях между ними.

1.2 Базы данных и СУБД

Понятие «база данных» (БД) можно применить практически к любой связанной между собой по определенному признаку информации, хранимой и организованной особым образом, чаще всего в виде набора таблиц. По сути БД – это некоторое подобие электронной картотеки или хранилища данных, расположенное в компьютере в виде одного или нескольких файлов [2].



***База данных** – поименованная совокупность взаимосвязанных данных, отображающая состояние объектов и их связей в некоторой предметной области и находящаяся под управлением специального программного комплекса.*

Очень часто базу данных называют информационной моделью некоторой предметной области, поскольку любая БД хранит в себе информацию определенного назначения одной из областей жизнедеятельности человека. Например, БД для отдела кадров, бухгалтерии, магазина, поликлиники, столовой и т. д. Базы данных в разных областях, несмотря на существенные различия в объектах и связях этих областей, организованы по одним и тем же правилам и предоставляют по сути одни и те же возможности при эксплуатации.

Рассмотрим ещё один часто употребляемый термин.



Программный комплекс, реализованный для сбора, хранения и обработки информации для принятия решений и реализации функций управления, называется информационной системой (ИС).

Несмотря на схожесть понятий «база данных» и «информационная система», важно понимать различия между ними. Зачастую информационная модель предметной области в дальнейшем реализуется именно в виде *автоматизированной информационной системы (АИС)*, основным компонентом которой является база данных. Например, заходя на сайт интернет-магазина, мы становимся пользователями информационной системы, которая большинство своих данных хранит в базе данных. Однако сам сайт назвать базой данных нельзя, поскольку для его функционирования используется также целый ряд других программных средств.

В любом случае при взаимодействии человека с информацией из БД всегда возникает необходимость в выполнении ряда обязательных операций, например:

- добавление новой информации;
- изменение информации;
- поиск информации;
- обработка информации;
- удаление информации и т. п.

Весь этот набор операций, а также процессы, связанные с созданием БД, её поддержкой и обеспечением доступа пользователей осуществляется с помощью специального программного инструментария – системы управления базами данных (СУБД) [2].



.....

***Система управления базами данных** – это совокупность программных и языковых средств, предназначенных для создания, ведения и совместного использования базы данных многими пользователями и обеспечения её взаимодействия с прикладными программами.*

.....

СУБД представляет собой инструмент сбора и хранения больших массивов информации и эффективного управления ими, позволяющий сохранять данные в целостности и безопасности на протяжении длительного времени.

В общем случае под СУБД можно понимать любой программный продукт, поддерживающие процессы создания и ведения БД [3]. Однако у любого программного продукта, претендующего на звание СУБД, должны быть реализованы функции, позволяющие пользователю взаимодействовать с базой данных, а также быть уверенным в сохранности и безопасности хранимых данных.

К базовым функциям СУБД относятся следующие [3–5]:

1. Средства постоянного хранения данных.

СУБД, как и файловые системы, поддерживают возможности хранения больших фрагментов данных, которые существуют независимо от каких бы то ни было процессов их использования. Однако СУБД значительно превосходят файловые системы в отношении гибкости представления информации, предлагая структуры, обеспечивающие эффективный доступ к большим порциям данных.

2. Поддержка языков доступа к данным.

СУБД позволяет пользователю или прикладной программе обращаться к данным и изменять их посредством команд развитого языка запросов.

3. Управление данными.

СУБД обеспечивает выполнение операций создания и манипулирования данными (выбор, вставка, обновление, удаление данных и т. п.) и одновременное отображение (выполнение) этих операций над физическими данными. Таким образом, пользователя или разработчика не должны волновать особенности расположения файлов БД и их организации на физическом носителе, эти функции в полной мере реализуются инструментами СУБД.

Помимо базовых функций СУБД также можно выделить ряд *дополнительных* функций, связанных с проблемами безопасности, целостности и эффективности функционирования СУБД:

1. Управление одновременным доступом к БД.

СУБД должна обеспечивать одновременный доступ к данным многих пользователей таким образом, чтобы действия, осуществляемые пользователями с объектами БД, не препятствовали друг другу и не нарушали целостности БД.

2. Обеспечение безопасности данных.

Под функцией безопасности (или физической защиты) данных подразумевается предотвращение разрушения или искажения данных в результате программного или аппаратного сбоя. Обеспечение безопасности во многих случаях является внутренней задачей СУБД, поскольку связано с её нормальным функционированием и решается на уровне компонентов СУБД. Цель восстановления базы данных после сбоя – обеспечение сохранности результатов всех операций в восстановленной БД и скорейшее возвращение к нормальному продолжению работы.

3. Обеспечение защиты от несанкционированного доступа.

Под функцией понимается защита данных от преднамеренного искажения и/или доступа пользователей или посторонних лиц. Рассматривая техническую сторону обеспечения защиты данных в БД от несанкционированного доступа, можно выделить общий принцип управления доступом к БД: СУБД не должна разрешать пользователю выполнение какой-либо операции над данными, если он не получил на это права.

4. *Обеспечение целостности данных.*

Эта функция подразумевает наличие средств, позволяющих удостовериться, что информация в базе данных всегда остается корректной и полной. Целостность данных должна обеспечиваться независимо от того, каким образом данные заносятся в память: в интерактивном режиме, посредством импорта или с помощью специальной программы. Используемые в настоящее время СУБД обладают встроенными средствами обеспечения целостности данных и безопасности.

5. *Сервисные и вспомогательные функции.*

Вспомогательные утилиты обычно предназначены для оказания помощи администратору в эффективном управлении СУБД. Некоторые утилиты работают на внешнем уровне, а потому они, в принципе, могут быть созданы самим администратором, тогда как другие функционируют на внутреннем уровне системы и потому должны быть предоставлены непосредственно разработчиками СУБД. Приведем некоторые примеры подобных функций:

- функции и утилиты импортирования, предназначенные для загрузки базы данных из различных файлов или других внешних источников, а также утилиты экспортирования, которые служат для выгрузки базы данных в файлы;
- средства мониторинга и статистического анализа, позволяющие оценить производительность или степень использования базы данных;
- инструменты реорганизации индексов, предназначенные для перестройки индексов и обработки случаев их переполнения;
- инструменты сборки мусора и перераспределения памяти для физического устранения удаленных записей с запоминающих устройств, объединения освобожденного пространства и перераспределения памяти в случае необходимости.

Конечно, описанные выше базовые и дополнительные функции СУБД имели несколько общий характер. На самом деле реальный объем функциональных возможностей, предлагаемых в некоторой конкретной СУБД, отличается от продукта к продукту. Например, в СУБД, предназначенной для персонального использования, может не поддерживаться параллельный совместный доступ, а управление режимом безопасности, поддержанием целостности данных и восстановлением будет присутствовать только в ограниченной степени, тогда как современные СУБД промышленного масштаба предлагают все перечисленные выше функциональные возможности и многое другое.

1.3 Классификация СУБД

Существует несколько основных критериев, по которым в настоящее время производится классификация СУБД [3–6]:

1. *По степени универсальности* различают СУБД:

- общего назначения;
- специализированные.

СУБД общего назначения – программные комплексы, предназначенные для выполнения всей совокупности функций, связанных с созданием, обновлением и эксплуатацией базы данных информационной системы.

СУБД общего назначения не ориентированы на какую-либо предметную область. Данным СУБД присущи развитые функциональные возможности, позволяющие решать практически любые задачи.

Специализированные СУБД создаются в тех случаях, когда ни одна из существующих СУБД общего назначения не может удовлетворительно решить задачи, стоящие перед разработчиками, например, не достигается требуемое быстродействие обработки или специфические функции конкретной предметной области требуют отдельного решения, плохо реализуемого стандартными функциями обычных СУБД. Специализированные СУБД предназначены для решения конкретной задачи, а необходимые параметры этого решения достигаются за счёт знания особенностей конкретной предметной области, а также путём сокращения функциональной полноты системы. К их числу можно отнести, например, информационно-поисковые или информационно-правовые системы. В частности, существует несколько известных информационно-правовых систем, таких как «Консультант», «Кодекс», «Гарант» и т. п., работающих под управлением специализированных СУБД, кроме того к этой группе можно отнести продукты платформы «1С», «Галактика» и т. д.

2. *По модели данных* используемой БД различают:

- иерархические;
- сетевые;
- реляционные;
- постреляционные;
- объектно-ориентированные;
- многомерные.

Этот критерий классификации более подробно рассмотрен в п. 1.5 учебного пособия.

3. По методам обработки и организации хранения данных различают СУБД:

- централизованные;
- распределенные.

Централизованная СУБД работает с базой данных, которая хранится в памяти одной вычислительной системы, к которой могут быть подключены несколько других компьютеров.

Для рассмотрения понятия «распределенная СУБД» сначала необходимо обратиться к термину «распределенная база данных».

Распределенная БД – это набор логически связанных между собой разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети, но воспринимаются пользователями как единая БД.

Распределенная СУБД – это программный комплекс, предназначенный для управления распределенными БД и позволяющий сделать распределённость информации прозрачной для конечного пользователя.

Система управления распределенными базами данных (СУРБД) состоит из единой логической базы данных, разделенной на некоторое количество фрагментов. Каждый фрагмент базы данных сохраняется на одном или нескольких компьютерах, которые соединены между собой линиями связи и каждый из которых работает под управлением отдельной СУБД.

4. По способу доступа к БД различают СУБД:

- файл-серверные;
- клиент-серверные;
- встраиваемые.

В *файл-серверных СУБД* файлы данных располагаются централизованно на файл-сервере. Функциональная часть СУБД, как правило, располагается на каждом клиентском компьютере, а доступ к данным осуществляется через локальную сеть. К преимуществам данной архитектуры относятся простота в реализации и использовании системы при небольшом количестве пользователей, а также низкая нагрузка на ЦП сервера. Недостатками являются потенциально высокая загрузка локальной сети, затруднённость централизованного управления, проблемы обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях и системах с ограниченным числом пользователей, которые используют функции управления БД. К этому классу относятся следующие СУБД: MS Access, FoxPro, dBase, Paradox и др.

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД для многих клиентов, находящихся в сети. В контексте базы данных клиентская часть СУБД управляет пользовательским интерфейсом и логикой приложения, действуя как интеллектуальная рабочая станция, на которой выполняются приложения баз данных. Клиент принимает от пользователя запрос, проверяет синтаксис и формирует запрос к базе данных на языке БД (например, SQL), который соответствует логике приложения. Затем он передает сообщение серверу, ожидает поступления ответа и форматирует полученные данные для представления их пользователю. Сервер принимает и обрабатывает запросы к базе данных, а затем передает полученные результаты обратно клиенту. В функции сервера также входят обработка, проверка полномочий клиента, обеспечение требований целостности, поддержка системного каталога, а также выполнение запроса и обновление данных. Кроме того, сервером поддерживается управление параллельностью и восстановлением, а также целый ряд других необходимых функций. Клиент-серверные СУБД в настоящее время занимают центральное место среди всех СУБД и являются основой при разработке большинства информационных систем с различным числом пользователей. К этому классу СУБД относятся MS SQL Server, Oracle, MySQL, PostgreSQL, Interbase, Firebird и др.

Встраиваемая СУБД – это библиотека, которая позволяет унифицированным образом хранить большие объёмы данных на локальной машине. В отличие от клиент-серверных встраиваемые СУБД не представляют собой независимый процесс, с которым взаимодействует программа, а являются составляющей частью программы и полностью либо частично интегрируются в нее. Встраиваемые СУБД зачастую превосходят по скорости клиент-серверные, в частности при использовании операций чтения, и при этом не требуют установки сервера, поэтому востребованы в локальном ПО. Зачастую встраиваемые СУБД ориентируются либо на решение узкого спектра специфических задач, либо создаются специально для использования в конкретном конечном продукте или работы с одним средством разработки. К этому классу СУБД относятся SQLite, Berkeley DB, Microsoft SQL Server Compact и др.

1.4 Представление данных в БД

Как уже говорилось, каждая информационная система в зависимости от назначения имеет дело с той или иной частью конкретного мира, которую принято называть *предметной областью*. Анализ предметной области является не-

обходимым начальным этапом разработки любой информационной системы. Именно на этом этапе определяются информационные потребности всех будущих пользователей системы, которые в дальнейшем и определяют содержание базы данных [7].

Объединяя частные представления о содержимом будущей базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в приложениях, разработчик сначала создает обобщенное неформальное описание создаваемой базы данных, называемое *инфологической моделью*. Одной из главных задач разработчика на начальных этапах проектирования является структуризация полученных пользовательских представлений и выявление объектов и взаимосвязей в предметной области на основе полученных данных.

В процессе научных исследований, посвященных тому, как именно должна быть устроена СУБД, предлагались различные способы реализации. Самым жизнеспособным из них оказалась трехуровневая архитектура баз данных, предложенная в 1978 г. комитетом планирования стандартов и норм SPARC (Standards Planning And Requirements Committee) американского института по стандартизации ANSI (American National Standard Institute), изображенная на рисунке 1.1.

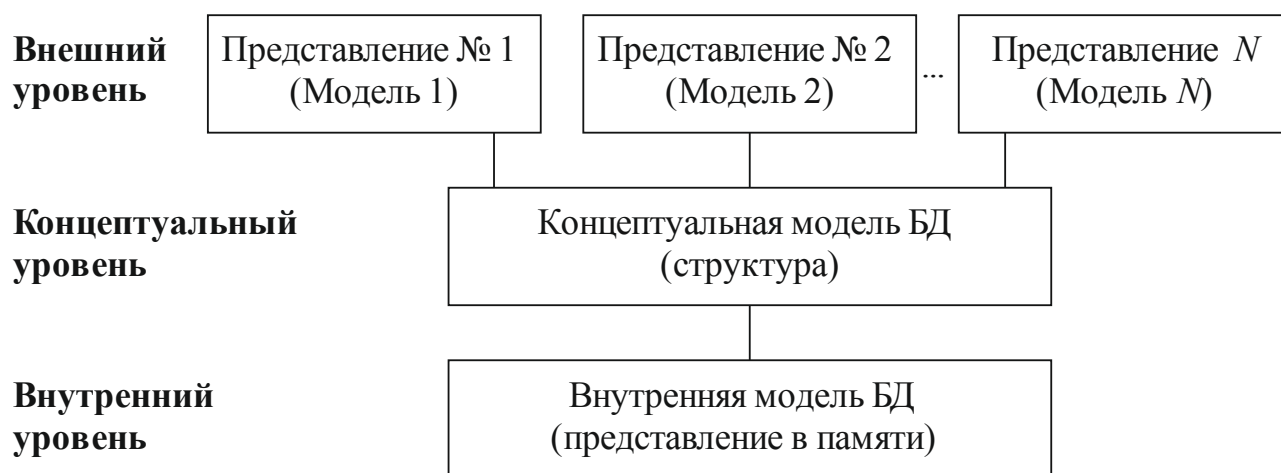


Рис. 1.1 – Трехуровневая архитектура ANSI/SPARC

Внешний уровень (уровень пользователей) – структурный уровень БД, определяющий пользовательские представления данных. Каждая пользовательская группа получает свое собственное представление данных в БД, так же, как и впоследствии каждое приложение или часть интерфейса ИС конкретного пользователя будет работать только с определенной частью данных. Каждое та-

кое представление данных дает ориентированное на пользователя описание элементов данных, из которых состоит представление данных, и отношений между ними. Совокупность таких пользовательских представлений данных и дает внешний уровень.

Концептуальный уровень (уровень проектировщика) – структурный уровень, определяющий логическую схему базы данных, поэтому данный уровень также часто называют логическим. На данном уровне выполняется концептуальное проектирование базы данных, которое включает анализ информационных потребностей пользователей и определение нужных им элементов данных. Результатом концептуального проектирования является концептуальная схема, логическое описание всех элементов данных и отношений между ними.

Внутренний уровень (уровень физических устройств) – это уровень, определяющий физический вид базы данных, ориентированный на физическое хранение и связанный со способами сохранения информации на физических устройствах. С данным уровнем связаны внешние запоминающие устройства, физические адреса, индексы, указатели и т. д. На этом уровне решается, какие физические устройства будут хранить данные, какие методы доступа будут использоваться для извлечения и обновления данных и какие меры следует принять для поддержания или повышения быстродействия системы управления базами данных.

Другими словами, можно сказать, что трехуровневая архитектура позволяет взглянуть на каждую конкретную базу данных с трёх различных сторон. Пользователь видит готовые таблицы и формы, осуществляя операции с базой данных через свой персональный интерфейс, которые может отличаться от интерфейса других пользователей. Разработчик или проектировщик работает со схемой баз данных и решает, каким образом организовать рациональное хранение данных и обеспечить выдачу необходимых данных пользователю по запросу. Третий уровень, по сути, связывает работу СУБД с файловой системой и аппаратными устройствами.

Независимость от данных. Основным назначением трехуровневой архитектуры является обеспечение независимости от данных, которая означает, что изменения на одном из уровней не должны оказывать влияние на другие уровни. Различают два типа независимости от данных: *логическую* и *физическую*.

Логическая независимость от данных означает полную защищенность внешних представлений от изменений, вносимых в концептуальную схему. Такие изменения концептуальной схемы, как добавление или удаление новых

сущностей, атрибутов или связей, должны осуществляться без внесения изменений в уже существующие внешние представления (схемы) или переписывания прикладных программ.

Физическая независимость от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему. Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю схемы. Пользователь может заметить изменения только в общей производительности системы [8].

1.5 Модели данных

Хранимые в базе данные имеют определенную логическую структуру, т. е. представляют собой набор объектов, определенным образом организованных и связанных друг с другом, а также правила управления этими объектами.



.....

Модель данных (МД) – это совокупность структур данных и операций над этими структурами, поддерживаемых конкретной СУБД.

.....

В качестве основных компонентов моделей данных выделяют: структуры данных, операции над данными, ограничения модели.

Как уже отмечалось, все СУБД по модели данных можно разделить на иерархические, сетевые, реляционные, постреляционные, многомерные и объектно-ориентированные.

1.5.1 Иерархическая модель

Иерархическая модель данных является наиболее простой среди всех моделей. Исторически она появилась первой и была реализована в СУБД IMS компании IBM – одной из первых зарегистрированных промышленных СУБД. Появление этой модели связано с тем, что в реальном мире очень многие отношения выстроены иерархическим способом, когда один объект является родительским, а с ним может быть связано некоторое количество подчиненных объектов, каждый из которых в свою очередь может выступать родительским для объектов следующего уровня.

Иерархическая древовидная структура данных строится из узлов и ветвей. *Узел* представляет собой совокупность атрибутов данных, описывающих некоторый объект. Наивысший узел в иерархической древовидной структуре называется *корнем*. Зависимые узлы располагаются на более низких уровнях дерева (рис. 1.2).

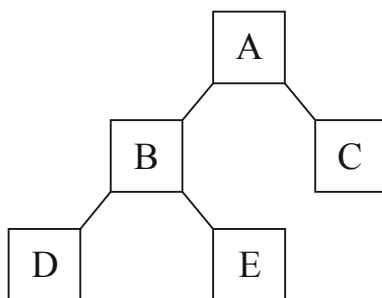


Рис. 1.2 – Пример структуры иерархического дерева

Иерархическая МД организует данные в виде иерархической древовидной структуры: каждый экземпляр корневого узла образует начало записи БД, т. е. иерархическая БД состоит из нескольких деревьев.

Иерархическая древовидная структура всегда удовлетворяет следующим условиям:

- 1) иерархия всегда начинается с корневого узла;
- 2) каждый узел состоит из одного или нескольких атрибутов, которые описывают объект в данном узле;
- 3) узел, находящийся на предшествующем уровне, является исходным для новых зависимых узлов. Зависимые узлы могут добавляться как в вертикальном, так и в горизонтальном направлении без всяких ограничений (исключение: на первом уровне может находиться только один узел, называемый корневым);
- 4) каждый узел, находящийся на нижнем уровне, соединен с одним и только одним узлом на уровне выше;
- 5) доступ к каждому узлу, за исключением корневого, происходит через исходный узел;
- 6) возможно существование любого числа экземпляров узлов каждого уровня. Каждый экземпляр некоторого узла В (за исключением корневого) соединен с экземпляром исходного узла, т. е. может существовать много экземпляров узла В. Для каждого экземпляра узла А может существовать нуль, один или несколько экземпляров порожденного им узла В и т. д. [8].

Иерархическая модель базы данных, описывающая структуру организации, представлена на рисунке 1.3.

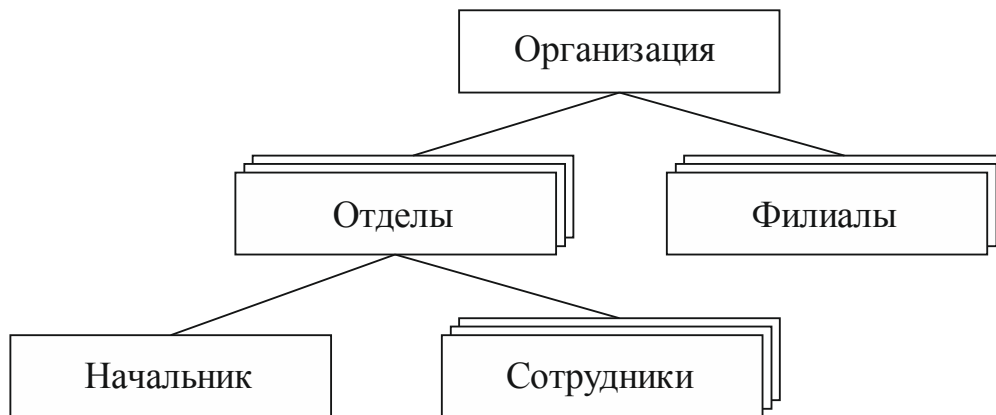


Рис. 1.3 – Пример иерархической модели БД

Достоинства модели:

1. Наличие хорошо зарекомендовавших себя СУБД, поддерживающих иерархические модели данных.
2. Простота понимания и удобство работы с иерархически упорядоченной информацией.

Недостатки модели:

1. Из-за строгой иерархической упорядоченности объектов модели значительно усложняются операции включения и удаления данных.
2. Доступ к любому узлу возможен только через исходный, что снижает скорость системы при ответах на запросы.
3. Сложность реализации неиерархических структур.

В настоящее время иерархическая модель данных практически не применяется и представляет интерес как одна из первых появившихся на свет моделей.

1.5.2 Сетевая модель

Стандарт сетевой модели данных впервые был разработан в 1975 г. организацией CODASYL (Conference of Data System Languages), которая определила базовые понятия модели и формальный язык описания [8].

Сетевая модель данных является развитием иерархической модели, позволяя отображать разнообразные взаимосвязи элементов данных в виде произвольного графа (рис. 1.4).

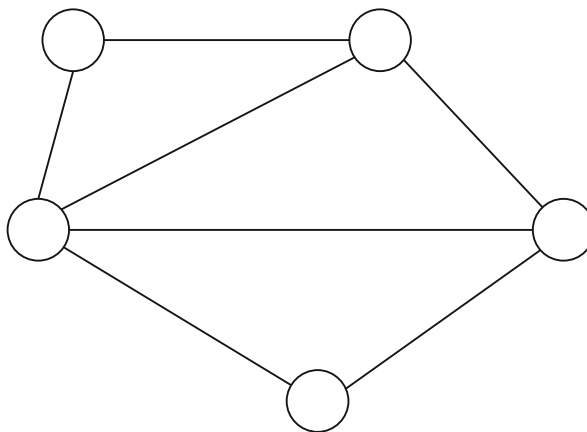


Рис. 1.4 – Вид связей в сетевой модели

В вершинах графа при интерпретации размещаются типы объектов (сущностей), а ребра графа являются типами связей между типами сущностей. Отличием сетевой модели от иерархической является то, что любая запись может входить в любое число именованных связей и фактически здесь нет корневого узла, т. к. любая запись может быть определена как точка входа.

Основные типы структур данных сетевой модели: элемент данных, агрегат, запись, набор, база данных.

Элемент данных – наименьшая поименованная единица данных (аналог поля в файловых системах).

Элементы данных обычно используются для представления отдельных элементарных свойств объектов, например <идентификационный_номер>, <код_студента> и т. п. Имя элемента данных используется для его идентификации в схеме структуры данного более высокого уровня.

Агрегат данных – поименованная совокупность элементов данных внутри записи, которую можно рассматривать как единое целое.

Запись – поименованная совокупность элементов данных или элементов данных и агрегатов.

Набор – поименованная совокупность записей, образующих двухуровневую иерархическую структуру. Этот термин не является аналогом набора файлов. Каждый тип набора представляет собой отношение (связь) между двумя или несколькими типами записей.

Достоинства модели

Главным достоинством сетевой модели является возможность простой реализации часто встречающихся в реальном мире взаимосвязей «многие-ко-многим».

Недостатки модели:

1. Высокая сложность и жесткость схемы БД.
2. Слабый контроль целостности связей из-за возможности установления произвольного количества связей в БД.
3. Возможная потеря независимости данных от программ при реорганизации БД.
4. Сложность для понимания и разработки прикладных программ на основе данной модели [3].

1.5.3 Реляционная модель

Реляционная модель данных была разработана в начале 1970-х гг. американским математиком Э. Ф. Коддом, который впервые сформулировал основные понятия и ограничения модели. Предложения Кодда были настолько эффективны для систем баз данных, что за разработку модели он был удостоен престижной премии Тьюринга в области теоретических основ вычислительной техники. Простота, гибкость и наглядность данной модели как для разработчиков, так и для всех пользователей, привлекли к ней широкое внимание и определили большую популярность. В связи с этим уже в 1980-х гг. реляционная модель стала промышленным стандартом СУБД [8].

Реляционная модель основывается на понятии *отношение* (от англ. – *relation*). Наглядной формой представления отношения является обычная двумерная таблица, которая имеет строки (записи) и столбцы (колонки).

Таблица отражает реальный или виртуальный объект реального мира, называемый сущностью, а каждая её строка отражает один конкретный экземпляр объекта – экземпляр сущности. Таблица содержит 2 главные части: заголовков и набор записей. При этом все записи обязательно имеют одинаковую структуру, описанную в заголовке таблицы. Например, таблица «Информация о сотрудниках», изображенная на рисунке 1.5, содержит сведения о сотрудниках, у каждого из которых имеются характеристики – номер, ФИО, должность и телефон.

Для организации хранения, обработки и поиска данных о различных объектах, а также реализации взаимосвязей в предметной области в таблицах (отношениях) используются так называемые ключевые поля.


Достоинства модели:

1. Простота и наглядность представления данных.
2. Наличие развитого языка управления данными.

3. Использование практически во всех современных СУБД.
4. Независимость прикладных программ от физической реализации БД.

Недостатки модели:

1. Сложность описания иерархических, сетевых связей, а также предметных областей со сложными структурами данных.
2. Относительно большое использование внешней памяти.



Наименование таблицы

Заголовок таблицы

Строки таблицы

Номер	ФИО	Должность	Телефон
1	Иванов Сергей Петрович	Инженер	1100
2	Петров Иван Сергеевич	Программист	1101

Рис. 1.5 – Представление данных в реляционной модели

Ещё раз отметим, что реляционная модель на сегодняшний день является основной для большинства существующих СУБД. Более подробно реляционная модель данных рассматривается в главе 3.

1.5.4 Объектно-ориентированная модель

Объектно-ориентированная модель данных начала разрабатываться в связи с появлением объектно-ориентированных языков программирования в 1990-е гг.

Стандартизированная объектно-ориентированная модель описана в рекомендациях стандарта ODMG-93 (Object Database Management Group). Реализовать в полном объеме данные рекомендации на настоящий момент не удалось, а потому можно говорить лишь об абстрактном «объектном» подходе к логическому представлению данных и разных объектно-ориентированных реализациях.

Объектно-ориентированную БД графически можно представить в виде дерева, узлами которого являются объекты. Свойства объектов описываются некоторым стандартным типом (например, string) или собственным типом пользователя (class). Значением свойств типа string является строка символов. Значение свойства типа class есть объект, являющийся экземпляром соответствующего класса. Каждый объект – экземпляр класса считается потомком объекта, в котором он определен как свойство. Объект – экземпляр класса принадлежит своему классу и имеет одного родителя. Родовые отношения в БД образуют связную иерархию объектов.

Структура объектно-ориентированного подхода описывается с использованием следующих понятий:

- *инкапсуляция* – ограничивает область видимости имени свойства пределами того объекта, в котором оно определено, то есть каждый объект имеет своё некоторое внутреннее состояние, а также набор методов, с использованием которых он получает доступ к чтению или изменению этого состояния;
- *наследование* – распространяет область видимости свойств объекта на всех его потомков. В данном случае имеется в виду возможность создания новых классов из старых, которые будут наследовать описание и методы своих предков, с возможностью некоторого их изменения;
- *полиморфизм* означает способность одного и того же программного кода работать с разнородными данными. Различные объекты получают возможность по-разному реагировать на одинаковые внешние события в зависимости от реализации их методов.

Достоинства модели:

1. Беспрепятственная интеграция между данными и их обработкой в приложениях.
2. Возможность отображения информации о сложных взаимосвязях объектов.
3. Возможность введения новых типов данных и операций с ними.

Недостатки модели:

1. Отсутствие общего языка манипулирования данными.
2. Низкая скорость выполнения запросов.
3. Проблемы реализации ограничений целостности данных [3].

1.5.5 Многомерная модель

Многомерная модель данных стала широко применяться с середины 1990-х гг., после того как Э. Кодд опубликовал в 1993 г. программную статью, содержащую основные требования к системам OLAP (OnLine Analytical Processing – оперативная аналитическая обработка) по представлению и обработке многомерных данных.

К этому времени стало очевидно, что реляционные модели данных несмотря на свою универсальность не совсем оптимальны при работе в системах интерактивной аналитической обработки. Для этих целей стали появляться многомерные СУБД.

Информация в многомерной модели представляется в виде многомерных массивов. В одной базе данных, построенной на многомерной модели, может храниться множество таких массивов, на основе которых можно проводить совместный анализ показателей. Если речь идет о многомерной модели с мерностью больше двух, то необязательно информация визуально представляется в виде многомерных объектов. В таком случае пользователь в качестве внешней модели данных получает для анализа определенные срезы или проекции кубов, как на рисунке 1.6, представляемые в виде обычных двумерных таблиц или графиков.

	2016		
	2015		
	2014		
Иванов	3,215	5,615	8,607
Петров	3,570	4,288	9,152
Сидоров	3,064	5,114	7,440
	Мониторы	Ноутбуки	Телефоны

Рис. 1.6 – Представление данных в многомерной модели

Основные понятия многомерных моделей: измерение и ячейка.

Измерение – это множество однотипных данных, образующих одну из граней гиперкуба. В многомерной модели измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба.

Ячейка – это поле, значение которого однозначно определяется фиксированным набором измерений. Тип поля чаще всего определен как цифровой. В зависимости от того, как формируются значения некоторой ячейки, она может быть переменной (значения изменяются и могут быть загружены из внешнего источника данных или сформированы программно) либо формулой (зна-

чения, подобно формульным ячейкам электронных таблиц, вычисляются по заранее заданным формулам).

Достоинства модели:

1. Удобство в работе с многомерными данными.
2. Эффективность аналитической обработки больших объемов данных.

Недостатки модели:

1. Узкоспециализированная модель.
2. Проигрывает другим моделям при решении стандартных задач [3].

1.5.6 Постреляционная модель

Постреляционная модель данных является расширенной реляционной моделью, в которой не действует ограничение неделимости данных в ячейках таблиц. Таким образом, постреляционная модель допускает многозначные поля, состоящие из набора значений. Набор значений фактически является отдельной таблицей, «вставленной» в ячейку основной таблицы.

Также данная модель поддерживает ассоциированные многозначные поля, совокупность которых называется *ассоциацией*. На длину и количество полей в записях таблицы не накладывается требование постоянства.

Достоинства модели:

1. Возможность представления модели в виде одной таблицы.
2. Высокая наглядность.
3. Повышенная эффективность обработки данных.

Недостатки модели:

1. Обеспечение целостности данных.
2. Обеспечение непротиворечивости данных [3].



Контрольные вопросы по главе 1

1. Дайте определение понятиям «данные» и «информация», поясните, как связаны эти термины между собой.
2. В чем отличия между информационной системой и базой данных?
3. Перечислите базовые функции СУБД.
4. Назовите основные критерии классификации СУБД.
5. Перечислите уровни архитектуры ANSI/SPARC.
6. Назовите недостатки многомерной модели данных.

2 Реляционная модель данных

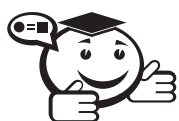
Основной моделью данных, используемой в большинстве современных СУБД, является реляционная модель. Для понимания основ организации данных в реляционной модели первоначально необходимо рассмотреть основные понятия и определения.

2.1 Определения модели



Реляционная модель данных (РМД) – это совокупность взаимосвязанных отношений, изменяющихся во времени, хранящая сведения о некоторой предметной области.

Основным понятием реляционной модели является термин «отношение», от которого и получила название данная модель. Дадим два определения данному термину.



Отношение – двумерная таблица, содержащая некоторые данные о предметной области.

Таким образом, отношение есть не что иное, как обычная таблица, состоящая из строк и столбцов. В дальнейшем встречаемый в данной главе термин «отношение» читатель для удобства может мысленно заменять термином «таблица». Однако важно запомнить, что не каждую таблицу, которую вы встречаете в литературе или при работе на компьютере, можно назвать отношением.

Если подходить к определению термина математически, то оно будет выглядеть следующим образом.



Отношение R – это подмножество кортежей $\langle d_1, d_2, \dots, d_n \rangle$, где $d_m \in D_m$. D_m – один из доменов отношения, а d_m – значение атрибута отношения, выбранное из домена.

Для однозначной интерпретации термина рассмотрим ещё несколько важных определений.



Сущность – произвольный объект реального мира, данные о котором необходимо хранить в базе данных.

Чаще всего сведения о сущности хранятся в отношении, а само отношение получает название сущности. Например, сведения о сущности «Студент» хранятся в отношении «Студент».



Атрибут – одно из свойств, характеризующих объект (сущность).

У каждого объекта (сущности) реального мира есть набор свойств, которые его характеризуют и позволяют отличать друг от друга. Например, у каждого студента есть фамилия и имя, номер группы, в которой он учится, дата рождения и т. д. Все эти характеристики и называются атрибутами. В отношении названия всех необходимых атрибутов объекта являются *заголовками столбцов*.



Кортеж – упорядоченный набор значений атрибутов.

В отношении все данные хранятся в виде набора строк, первая из которых представляет собой заголовочную часть, т. е. набор названий атрибутов, а все последующие хранят непосредственно сведения о конкретных экземплярах сущности. Здесь также необходимо отметить разницу между понятиями «сущность» и «экземпляр сущности». Сущности – это объекты, состоящие из набора атрибутов, т. е. «Студент», «Сотрудник», «Предмет» и т. д. А экземпляр сущности – это один из конкретных представителей данной сущности. Например, «Иванов Иван из гр. 110» и «Петров Петр из гр. 598» являются двумя экземплярами сущности «Студент». Сведения о каждом экземпляре сущности хранятся в строке таблицы, в РМД называемой кортежем.



Домен – множество всех возможных значений атрибута отношения.

Каждый домен образует значения одного типа данных. Например, домен для атрибута «Номер группы» будет состоять из всех возможных номеров

групп данного вуза. Соответственно, тип данных для атрибута «Номер группы» будет одинаковым для всех возможных значений, например числовым или текстовым.



Схема отношения – список атрибутов сущности.

Схему отношения, по сути, образует первая строка таблицы, т. е. её заголовочная часть.

Для удобства рассмотренные основные определения терминов реляционной модели приведены в таблице 2.1.

Таблица 2.1 – Основные элементы РМД

	Элемент	Представление
1.	Сущность	Объект реального мира
2.	Отношение	Таблица
3.	Атрибут	Заголовок (название) столбца таблицы
4.	Схема отношения	Строка заголовков таблицы (первая строка)
5.	Кортеж	Строка таблицы (запись таблицы)
6.	Домен	Множество возможных значений атрибута

В качестве примера рассмотрим отношение «Студент» из таблицы 2.2, состоящее из 4 кортежей, хранящее общую информацию о студентах вуза.

Таблица 2.2 – Отношение «Студент»

Номер	ФИО	Дата рождения	Группа
1	Иванов Сергей Петрович	14.05.1988	116
2	Петров Иван Сергеевич	27.01.1989	598
3	Алексеев Семен Олегович	05.10.1993	445
4	Белов Петр Иванович	01.11.1988	116

Отношение «Студент» хранит в себе информацию о сущности (объекте) «Студент». Сущность «Студент» имеет 4 атрибута: «Номер», «ФИО» (фамилия, имя, отчество), «Дата рождения» и «Группа». Отношение состоит из 4 кортежей и хранит информацию о 4 экземплярах сущности – это студенты Иванов, Петров, Алексеев и Белов. Заголовочная строка таблицы, как уже упоминалось,

называется схемой отношения, в данном примере она выглядит так – Студент (Номер, ФИО, Дата рождения, Группа). Каждый из кортежей также можно называть строками или записями таблицы, а значения каждого атрибута в записи – значениями полей записи. Порядок кортежей в отношении в общем случае не определен, хотя в данном примере для удобства все записи отсортированы по значению первого атрибута.

Подводя итог данному разделу, сформулируем окончательно правила, позволяющие считать таблицу отношением:

- все строки таблицы обязательно имеют *одинаковую структуру*, т. е. порядок следования значений атрибутов в полях идентичен во всей таблице;
- в таблице не может быть двух атрибутов с одинаковыми *названиями*;
- значение любого поля должно быть *простым*, т. е. недопустима группа значений в одном поле;
- в таблице *не должно быть* двух и более абсолютно *одинаковых* строк;
- размещение строк в таблице *произвольно*.

2.2 Ключевые атрибуты

Одним из важных свойств отношения является возможность различать экземпляры сущностей, таким образом, важно понимать, что *в отношении не должно быть двух одинаковых кортежей*. Допустимо наличие одинаковых значений отдельных полей, например, несколько студентов обучаются в одной группе или родились в один и тот же день. Теоретически возможна даже ситуация, что два студента будут иметь одинаковые фамилию, имя и отчество, а также дату рождения и будут обучаться в одной группе. Такую ситуацию представить крайне сложно, но такое всё-таки может произойти. Как в такой ситуации поступит преподаватель, чтобы при обращении к нужному студенту не отвлекался второй? Придется искать дополнительные отличительные признаки, например «светлый»/«темный» или просто «первый»/«второй» и т. д. Таким образом, потребуется наличие новых дополнительных атрибутов сущности. Аналогично при проектировании отношения важно предупреждать возможные ситуации дублирования кортежей и предусмотреть условия исключения таких ситуаций.

Первичный ключ

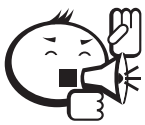
В реляционной модели для разрешения ситуаций дублирования данных и идентификации каждой записи используется элемент, называемый *первичным ключом (ПК)*.



*Первичный ключ (Primary Key, PK) – атрибут или набор атрибутов отношения, однозначно идентифицирующий любой из его кортежей. Первичный ключ – **составной**, если он состоит из двух или более атрибутов.*

В таблице 2.2 в качестве первичного ключа можно выбрать, например, атрибут «ФИО». Хотя, как уже говорилось, это может быть допустимо только при работе с небольшим набором записей, когда ситуация дублирования данного поля практически исключена. Для удобства чаще всего в таких ситуациях вводят так называемый *суррогатный*, или *искусственный*, первичный ключ, например «Номер», «Номер студента», «Код», «Код студента» и т. п. Зачастую такой ключ является ещё и счетчиком записей, присваивая каждой следующей добавляемой записи очередной порядковый номер. Данный ключ позволяет решить проблему дублирования записей, ведь даже при наличии ситуации, когда все поля двух записей будут совпадать, порядковый номер у таких практически одинаковых записей будет различным. Однако не стоит для каждого отношения в базе данных обязательно вводить искусственный первичный ключ, особенно это касается таблиц, которые связаны с другими.

Как уже отмечалось, в отношении не может существовать двух одинаковых кортежей. А значит, любой из них можно отличить от другого по одному или нескольким атрибутам (полям).



Каждое отношение обязательно имеет комбинацию атрибутов, которую можно использовать в качестве первичного ключа. В крайнем случае ключом отношения будут служить абсолютно все его атрибуты.

Во многих случаях в отношении можно определить несколько комбинаций атрибутов в качестве возможных ключей, каждую из которых можно выбрать в качестве первичного ключа. Все остальные ключи, кроме выбранного первичного, называют *альтернативными* или *возможными* ключами отноше-

ния. Если первичный ключ состоит из минимального набора атрибутов, то он *не избыточный*.

Функции первичного ключа:

1. *Исключение дублирования записей*. Значение первичного ключа должно быть уникально. В случае, когда первичный ключ составной значения отдельных атрибутов ключа могут совпадать, однако комбинация всех атрибутов должна быть уникальна. В качестве примера в таблице 2.3 приведено отношение «Ведомость», в котором первичным ключом служит комбинация атрибутов (Студент, Предмет). В данном отношении отдельные атрибуты ключа могут дублироваться, ведь один и тот же предмет могут сдавать многие студенты и каждый студент обычно сдает более одного предмета. Однако в таблице ни в коем случае не должна встречаться ситуация, когда один и тот же студент получил две оценки по одному и тому же предмету. Если при проектировании необходимо будет учесть ситуации возможной сдачи одним студентом одного и того же предмета несколько раз, тогда в состав первичного ключа необходимо дополнительно добавить дату сдачи экзамена.

Таблица 2.3 – Отношение «Ведомость»

Студент (ПК)	Предмет (ПК)	Дата сдачи экзамена	Оценка
Иванов Сергей Петрович	Базы данных	14.06.2016	4
Петров Иван Сергеевич	Базы данных	14.06.2016	5
Иванов Сергей Петрович	Информатика	18.06.2016	4
Петров Иван Сергеевич	Информатика	18.06.2016	4

Отметим также, что за ситуацию дублирования значений первичного ключа отвечает сама СУБД, и в случае возникновения подобных ситуаций процессор СУБД выдаст соответствующее сообщение, например о невозможности добавления новой записи.

2. *Индексирование и сортировка*. Для каждого первичного ключа по умолчанию создается соответствующий индекс, позволяющий ускорять в дальнейшем поиск записей в базе данных, а также производить сортировку записей по первичному ключу. Как уже отмечалось выше, очень часто первичный ключ дополнительно снабжен функцией счетчика, т. е. каждая следующая добавляемая в отношение запись автоматически получает следующее по порядку значение в поле первичного ключа.

3. *Связывание таблиц* – это важнейшая функция первичного ключа, позволяющая организовать связь между двумя таблицами. Однако для изучения данной функции рассмотрим сначала термин «внешний ключ».

Внешний ключ

Связывание таблиц помимо термина «первичный ключ» неотъемлемо включает в себя и понятие «внешний ключ» (ВК), которое также играет одну из важнейших ролей в реляционной модели данных.



***Внешний ключ** (Foreign Key, FK) – это атрибут или набор атрибутов отношения, значения которого являются значениями первичного ключа другого отношения, связанного с этим.*

С помощью первичных и внешних ключей в реляционной модели производят связывание отношений друг с другом. Это делается прежде всего для того, чтобы наиболее рациональным образом разместить информацию в базе данных. Рассмотрим организацию связывания отношений на рисунке 2.1.

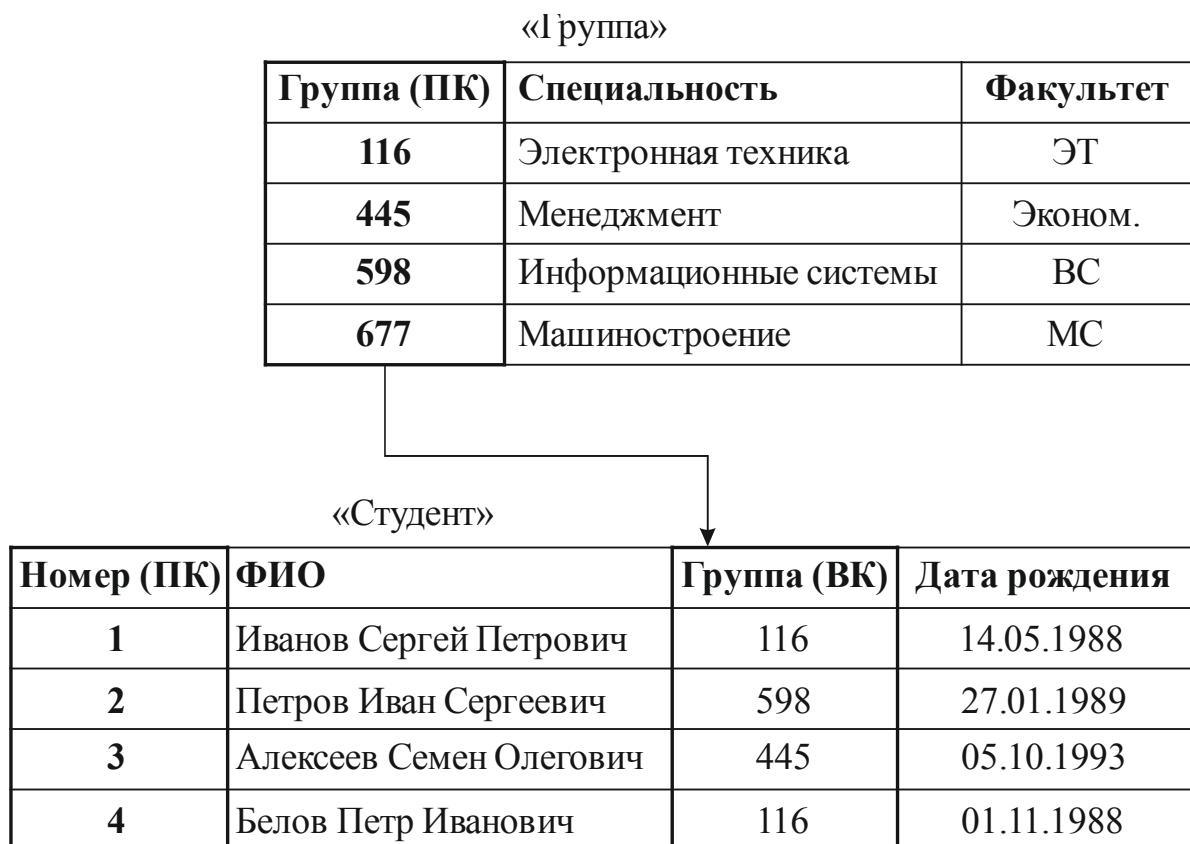


Рис. 2.1 – Связывание отношений «Студент» и «Группа»

В данном примере приведено связывание двух отношений – «Студент» и «Группа». Первичным ключом в отношении «Студент» является атрибут «Номер», а в отношении «Группа» – атрибут «Группа». Кроме того, в отношении «Студент» имеется ещё один выделенный атрибут – атрибут «Группа». Как видно из рисунка, значения данного атрибута являются значениями первичного ключа отношения «Группа», следовательно, согласно определению, данный атрибут и является внешним ключом.

На внешние ключи при связывании накладываются некоторые ограничения, например для каждого значения внешнего ключа в таблице должна обязательно присутствовать так называемая связанная запись в исходной таблице с соответствующим значением. Иными словами, в приведенном примере в отношении «Студент» не должно быть записи о студенте с номером группы, которая отсутствует в отношении «Группа».

2.3 Связи в БД

В любой предметной области отдельные объекты или сущности не существуют сами по себе. В большинстве случаев они связаны с другими объектами, а эти связи строятся на основе простых глагольных фраз. Например, «Студент Иванов обучается в группе 116» или «Студент Иванов изучает информатику». Вид связи между сущностями можно определить, сопоставив, какое количество экземпляров одной сущности связано или зависит от одного или нескольких экземпляров другой сущности.

Как уже отмечалось выше, связи между отношениями в самой базе данных реализуются с помощью первичных и внешних ключей. Далее будут рассмотрены следующие существующие взаимосвязи между объектами предметной области и их реализация в реляционной модели данных:

- один-к-одному;
- один-ко-многим;
- многие-ко-многим.

2.3.1 Связь «один-к-одному»

Самый простой вид связи – один-к-одному, обозначается как 1:1.



Связь «один-к-одному» означает, что каждому экземпляру первой сущности соответствует только один экземпляр второй

сущности, и наоборот, каждому экземпляру второй сущности соответствует только один экземпляр первой.

.....

Также возможен вариант, когда для некоторых экземпляров сущности может не существовать связанного экземпляра другой сущности, в таком случае связь называется необязательной.

Для реализации связи 1:1 между таблицами в базе данных необходимо произвести связывание их первичных ключей.

В качестве примера рассмотрим две сущности: «Студент» и «Паспорт», реализованные в виде таблиц БД на рисунке 2.2. Очевидно, что связь между двумя сущностями является связью «один-к-одному», поскольку у одного студента может быть только один паспорт, и у каждого паспорта имеется только один хозяин.

Номер (ПК)	ФИО	Группа	Дата рождения
1	Иванов Сергей Петрович	116	14.05.1988
2	Петров Иван Сергеевич	598	27.01.1989
3	Алексеев Семен Олегович	445	05.10.1993
4	Белов Петр Иванович	116	01.11.1988

↓

Номер (ПК)	Серия	Номер паспорта	Дата выдачи
1	6903	234235	18.05.2013
2	6904	267323	29.01.2014
3	6903	432567	08.10.2013
4	6905	234561	17.11.2015

Рис. 2.2 – Пример связи 1:1 между таблицами

В данном случае необходимо произвести связывание первичных ключей обоих отношений между собой.

На самом деле с точки зрения правильной организации базы данных хранить данные двух отношений со связью 1:1 целесообразнее в единой большой таблице, нежели в двух предлагаемых. Это позволит сэкономить место в памяти, а также упростит работу с запросами к БД.

Таким образом, в рассматриваемом примере связь «один-к-одному» между сущностями «Студент» и «Паспорт» в большинстве случаев логично было бы реализовать в виде одной таблицы со всеми атрибутами обеих сущностей, как в таблице 2.4. В качестве первичного ключа объединенной сущности можно выбрать любой из двух ключей двух отношений. В дальнейшем с помощью стандартных запросов к базе данных пользователь может сформировать исходные отношения «Студент» и «Паспорт», попросту исключив из запроса ненужные атрибуты.

Таблица 2.4 – Отношение «Студент»

Номер (ПК)	ФИО	Группа	Дата рождения	Серия	Номер паспорта	Дата выдачи
1	Иванов Сергей Петрович	116	14.05.1988	6903	234235	18.05.2013
2	Петров Иван Сергеевич	598	27.01.1989	6904	267323	29.01.2014
3	Алексеев Семен Олегович	445	05.10.1993	6903	432567	08.10.2013
4	Белов Петр Иванович	116	01.11.1988	6905	234561	17.11.2015

Тем не менее, существуют некоторые причины, по которым единый набор данных со всеми атрибутами не всегда объединяют в одну большую таблицу, а разбивают на две со связью «один-к-одному». В случае необходимости защитить конфиденциальную информацию от части пользователей может возникнуть необходимость разбить предлагаемые данные на 2 таблицы с различным уровнем доступа к ним разных пользователей. В некоторых ситуациях таблицы разбивают на несколько частей для ускорения доступа к наиболее часто используемым в работе данным. Кроме того, в некоторых случаях организации данных, когда единая таблица может содержать большое количество пустых строк, разделение может приводить и к экономии памяти.

2.3.2 Связь «один-ко-многим»

Данный вид связи обозначается 1:M.



Связь «один-ко-многим» означает, что каждому экземпляру первой сущности может соответствовать несколько экземпляров

второй сущности, а каждому экземпляру второй сущности соответствует только один экземпляр первой сущности.

.....

В таком случае первую сущность называют *родительской*, а вторую – *дочерней*.

Пример связи 1:M подробно рассмотрен на рисунке 2.3. Очевидно, что сущности «Студент» и «Группа» связаны друг с другом следующим образом:

- в каждой группе может обучаться много студентов;
- каждый студент может обучаться только в одной группе.



Рис. 2.3 – Пример связи 1:M между таблицами

Получается, что одному экземпляру сущности «Группа», например группе 116, будет соответствовать несколько экземпляров сущности «Студент», т. е. студенты Иванов и Белов. Согласно определению этот вид связи является связью 1:M. Сущность «Группа» выступает здесь в качестве родительской сущности, а сущность «Студент» – дочерней. Отметим, что каждая сущность может выступать и в роли дочерней, и в роли родительской одновременно, так как может одновременно быть связанной со многими сущностями (рис. 2.3).

В базе данных связь 1:M выполняется с помощью первичных и внешних ключей, когда поле, являющееся первичным ключом родительской сущности,

связывается с неключевым атрибутом дочерней сущности, который и становится внешним ключом. В базах данных связь вида 1:M является самой распространенной.

2.3.3 Связь «многие-ко-многим»

Данный вид связи обозначается как M:M или M:N.



Связь «многие-ко-многим» означает, что каждому экземпляру первой сущности может соответствовать несколько экземпляров второй сущности, а каждому экземпляру второй сущности может соответствовать несколько экземпляров первой сущности.

В качестве примера рассмотрим связь между сущностями «Группа» и «Предмет». Очевидно, что у каждой учебной группы имеются занятия по разным предметам, т. е. одному экземпляру сущности «Группа» соответствует несколько экземпляров сущности «Предмет». Кроме того, по каждому предмету занятия могут проходить у разных групп, т. е. одному экземпляру сущности «Предмет» также соответствует несколько экземпляров сущности «Группа». На рисунке 2.4 в качестве примера связи M:N изображено два отношения «Группа» и «Предмет», описывающие данные сущности и хранящие информацию об экземплярах. В данном случае приведена ситуация, когда у группы 116 есть занятия по предметам «Экономика» и «Физика», а предмет «Базы данных» изучается группами 445 и 598.



Рис. 2.4 – Пример связи M:N между сущностями

Однако на представленном рисунке нет реализации самой связи между отношениями с помощью первичных и внешних ключей, а указаны лишь записи, которые должны быть связаны по смыслу.

В реляционной модели для реализации связей такого вида используют дополнительную таблицу, в которой, по сути, и указывается – какой экземпляр

одной сущности связан с каким экземпляром второй сущности. На рисунке 2.5 приведено решение, позволяющее реализовать подобную ситуацию.



Рис. 2.5 – Реализация связи M:N в БД

В третьей таблице происходит сопоставление экземпляров обеих сущностей друг с другом. При этом каждая из исходных таблиц является родительской для третьей и связана с ней связью 1:M, а первичный ключ каждой из родительских таблиц становится частью составного ключа в новой подчиненной таблице. Более подробно особенности реализации связей в БД будут рассмотрены в главе 5.

2.3.4 Управление связанными записями в БД

Среди рассмотренных видов связи в базе данных в большинстве случаев используется только связь 1:M. Связь 1:1 используется редко, ведь чаще всего проектировщик объединяет атрибуты сущностей в единое отношение. Связь M:N также преобразовывается в 2 связи типа 1:M. Таким образом получается, что практически все связи в спроектированной базе данных относятся к виду 1:M.

При связывании таблиц любая СУБД в дальнейшем производит так называемый *контроль целостности данных*. Он включает в себя следующие правила:

- каждому экземпляру (записи) родительской таблицы может соответствовать ноль, одна или более записей дочерней таблицы;
- каждому экземпляру дочерней таблицы соответствует только один экземпляр родительской таблицы;
- в дочерней таблице не может быть экземпляров без соответствующего экземпляра в родительской.

Данные правила распространяются на все операции с данными в дочерних и родительских таблицах и позволяют избежать ситуаций потери или искажения данных, нарушающих полноту и целостность БД. Во многих случаях проектировщик имеет возможность настраивать подобные правила для различных ситуаций или, например, отключать контроль целостности, но тогда ответственность за корректное содержимое в базе данных будет нести он сам.

Рассмотрим наиболее типичные ситуации при операциях с данными.

При *добавлении* новых записей в таблицы БД важно понимать, что необходимо первоначально добавлять записи в родительскую таблицу, а затем – в дочернюю. В обратном случае возникнет ситуация, когда добавляемые записи не будут иметь связанных записей, а потому их добавление может быть просто заблокировано.

Изменение данных в записях связанных таблиц также контролируется СУБД. Если поля не участвуют в связях, т. е. не являются ни первичными, ни внешними ключами, то здесь происходит стандартный контроль, например, на соответствие типу данных. Если поле является внешним ключом в дочерней таблице, важно, чтобы после изменения его значения у него по-прежнему осталась связанная запись в родительской таблице. Другими словами, *дочерняя запись может менять «родителя», но остаться без него она не должна*.

В ситуации изменения первичного ключа родительской таблицы возможны следующие ситуации:

- если у записи родительской таблицы нет соответствующих записей в дочерней таблице, тогда изменение можно производить;
- если у записи родительской таблицы есть соответствующие записи в дочерней таблице, тогда изменение будет заблокировано процессором СУБД, т. к. в противном случае связанные записи в дочерней таблице будут связаны с несуществующей записью;
- если у записи родительской таблицы есть соответствующие записи в дочерней таблице, тогда изменение можно произвести в случае, когда включено так называемое *каскадное изменение данных*. Это значит,

что новое значение первичного ключа родительской таблицы будет присвоено всем внешним ключам связанных записей дочерней таблицы вместо старого измененного значения.

Удаление записей дочерней таблицы можно производить по своему усмотрению, не обращая внимание на записи родительской таблицы. Очевидно, что запись родительской таблицы может не иметь ни одной связанной записи в дочерней таблице.

В обратном случае, т. е. при удалении записи в родительской таблице, возникает ситуация, аналогичная той, которая происходит при изменении данных:

- если у записи родительской таблицы нет соответствующих записей в дочерней таблице, тогда удаление можно производить;
- если у записи родительской таблицы есть соответствующие записи в дочерней таблице, тогда удаление будет заблокировано процессором СУБД, т. к. в противном случае связанные записи в дочерней таблице будут связаны с несуществующей записью;
- если у записи родительской таблицы есть соответствующие записи в дочерней таблице, тогда удаление можно произвести в случае, когда включено так называемое *каскадное удаление данных*. Это значит, что вместе с удалением записи родительской таблицы произойдет удаление всех связанных записей дочерней таблицы.

2.4 Реляционные СУБД

В настоящее время большинство из представленных на рынке СУБД используют реляционную модель данных. Наиболее популярными из них являются MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Interbase, Firebird, Microsoft Access и т. д. В данном пособии особое внимание уделено СУБД Microsoft Access и MySQL как наиболее доступным и простым при первоначальном использовании. Данные продукты позволят на практике закрепить представленные в пособии теоретические основы баз данных.

2.4.1 Microsoft Access

СУБД Microsoft Access является системой управления реляционной базой данных, включающей все необходимые инструментальные средства для создания локальной базы данных, общей базы данных в локальной сети с файловым сервером или базы данных на SQL-сервере, а также для создания приложения

пользователя, работающего с этими базами данных. Первая версия программы вышла в 1992 г., а начиная с 1994 г. и по сей день разработанный на тот момент Access 2.0 неизменно входит в состав пакета Microsoft Office разных годов выпуска. На рисунке 2.6 изображено приложение Microsoft Access 2007.

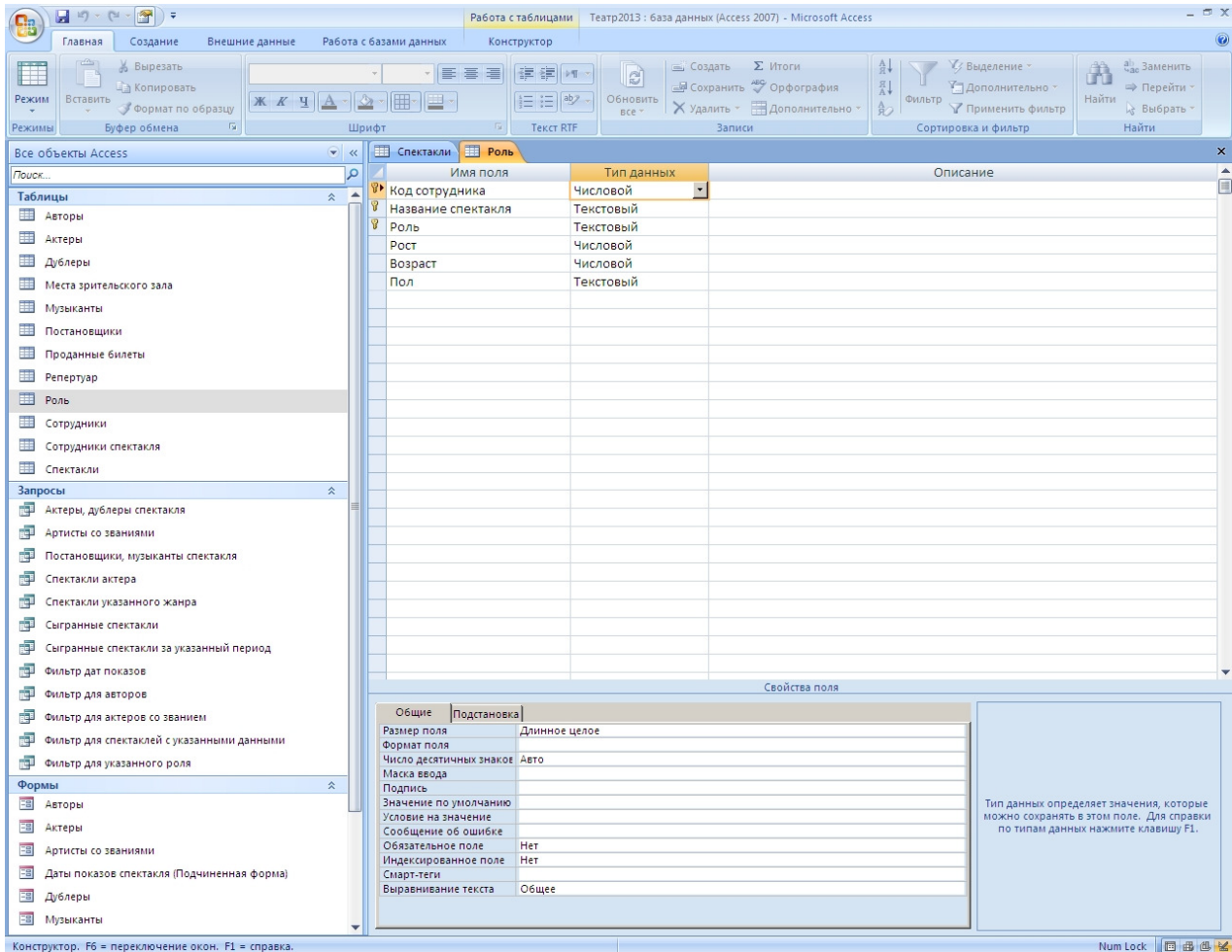


Рис. 2.6 – Microsoft Access 2007

MS Access относится к числу файл-серверных СУБД, поскольку всю информацию и объекты баз данных хранит в файле, к которому в случае необходимости обращаются другие пользователи и программы. База данных Access, создаваемая на локальном компьютере, хранит в файле не только все таблицы с данными, но и объекты приложения и интерфейса – формы, отчеты, а также программный код. Благодаря этому можно создать целую информационную систему, целиком хранящуюся в одном файле, что существенно упрощает как создание, так и распространение приложений баз данных.

Отличия Microsoft Access от большинства современных СУБД состоит в том, что данная СУБД предназначена для работы с различными группами пользователей: от начинающих разработчиков с минимальным уровнем знаний по

базам данных до профессиональных программистов и проектировщиков. С помощью мастеров и графических инструментов Access даже пользователи, не владеющие специальными навыками, могут весьма успешно разрабатывать приложения баз данных. В то же время, используя функциональные возможности Access, специалисты отделов информационных технологий смогут обеспечить разработку и поддержку приложений Access, создаваемых на уровне компаний.

СУБД обладает мощным и удобным аппаратом распределения данных по нескольким таблицам, реализуя практически любую модель данных предметной области. При этом обеспечивается совместная обработка и корректировка данных средствами форм и отчетов, обеспечивающими эффективный доступ и представление данных. В основе этих средств лежит реляционный язык структурированных запросов SQL.

Готовые приложения, доступные через шаблоны, поставляемые вместе с Access или на веб-сайтах Microsoft, позволяют пользователю, обладающему ограниченными знаниями в области баз данных, познакомиться с самыми важными функциями и во многих случаях начать работу с первого дня использования MS Access. Стандартные приложения из шаблонов можно использовать без каких-либо модификаций или настройки в режиме заполнения информацией.

Кроме создания новых таблиц в базе данных Access позволяет организовать связи с внешними данными с помощью интерфейса ODBC (Open Database Connectivity). Связанные таблицы отображаются и функционируют аналогично собственным таблицам. Через связанные таблицы возможно взаимодействие и со всеми распространенными серверами реляционных баз данных, включая Microsoft SQL Server, Oracle, IBM DB2, Informix и Sybase.

Интерактивные средства обеспечивают простоту разработки форм и отчетов. Режим представления форм и отчетов – режим макета – позволяет легко настроить их в соответствии с потребностями пользователя, наблюдая изменения этих объектов в реальном времени. Простые в использовании и разнообразные средства фильтрации данных позволяют, не перестраивая макет, получить отчеты, представляющие данные в самых разных разрезах, и использовать их для анализа данных.

К недостаткам СУБД MS Access относятся все недостатки, характерные для файл-серверных СУБД, а именно невозможность одновременной работы с большим количеством пользователей, слабая реализация функций надежности

и безопасности, а также практическое отсутствие реализации транзакций. По этим причинам СУБД MS Access чаще всего используется на небольших предприятиях с малым числом пользователей, а также в учебных целях.

Инструментальные средства MS Access

СУБД Access включает разнообразные инструментальные средства, ориентированные на создание объектов базы данных и приложений пользователя.

Средства графического конструирования позволяют создавать объекты базы данных и объекты приложения с помощью многочисленных графических элементов, не прибегая к программированию.

Разнообразные мастера в режиме ведения диалога с пользователем позволяют создавать объекты и выполнять разнообразные функции по реорганизации и преобразованию баз данных.

Среди средств графического конструирования и диалоговых средств следует выделить средства для создания:

- таблиц и схем баз данных, отображающих их связи;
- запросов выборки, отбирающих и объединяющих данные нескольких таблиц;
- виртуальную таблицу, которая может использоваться во многих задачах приложения;
- запросов на изменение данных базы;
- экранных форм, предназначенных для ввода, просмотра и обработки данных в диалоговом режиме;
- отчетов, предназначенных для просмотра и вывода на печать данных из базы и результатов их обработки в удобном для пользователя виде.

Средства программирования СУБД включают язык структурированных запросов SQL, язык макрокоманд и язык объектно-ориентированного программирования для приложений Microsoft Visual Basic for Applications (VBA). VBA является частью семейства Microsoft Visual Basic, которое входит в состав Visual Studio.

VBA является базовым компонентом Microsoft Office: он интегрирован в Access, Excel, FrontPage, Outlook, PowerPoint и Word. Все эти приложения, в том числе и локализованные на русском языке, используют англоязычный вариант VBA (включая справочную систему). VBA представляет собой базовую платформу программирования не только в среде Microsoft Office, но и многих

других приложений. VBA содержит средства доступа как к базам данных Access, так и к базам данных клиент-серверной архитектуры, таким как Microsoft SQL Server, Oracle и др.

Система доступа к данным, начиная с Access 2007, построена на основе ядра базы данных Access Database Engine, заменившего прежнюю версию ядра Microsoft Jet 4.0. Ядро базы данных выполняет загрузку, сохранение и извлечение данных в пользовательских и системных базах данных. Обеспечивает высокую производительность и улучшенные сетевые характеристики, поддержку двухбайтового представления символов Unicode, позволяющего использовать символы нескольких национальных алфавитов.

Схема данных

В СУБД Access процесс создания реляционной базы данных включает создание *схемы данных*. Схема данных на рисунке 2.7 наглядно отображает логическую структуру базы данных: таблицы и связи между ними, а также обеспечивает использование установленных в ней связей при обработке данных.

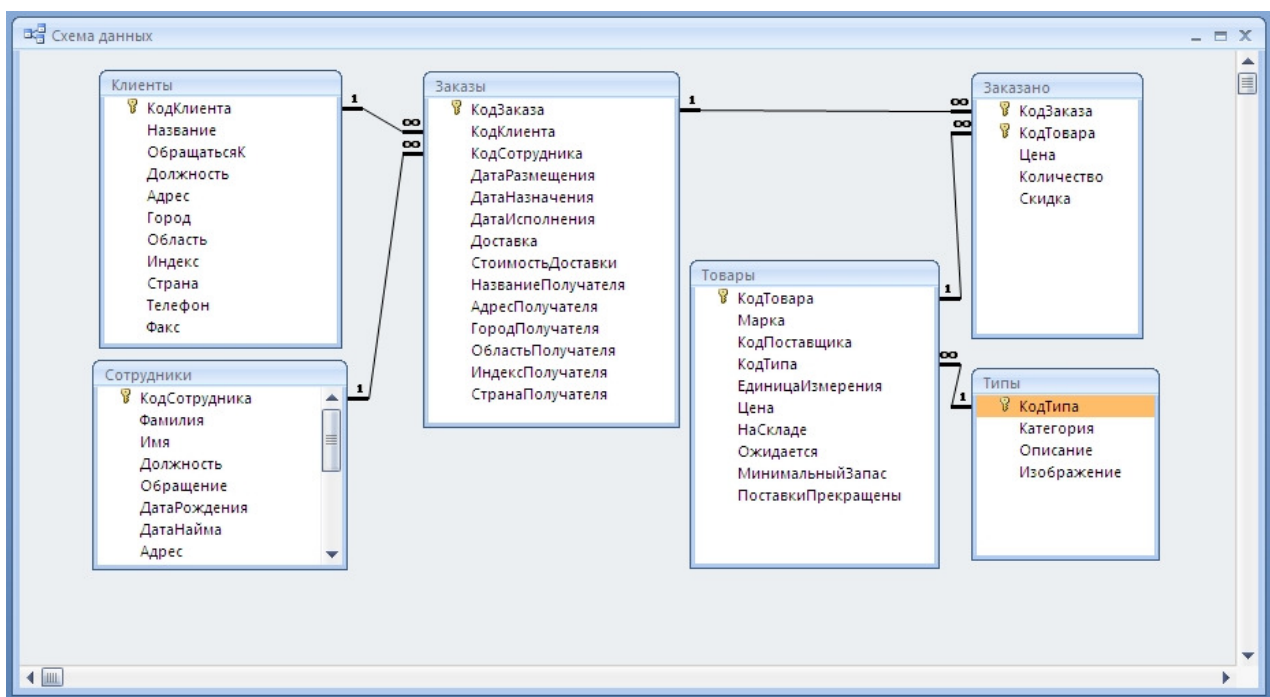


Рис. 2.7 – Пример схемы данных MS Access

Для нормализованной базы данных, основанной на одно-многочисленных и одно-однозначных отношениях между таблицами, в схеме данных для связей

таких таблиц по первичному ключу или уникальному индексу главной таблицы могут устанавливаться параметры обеспечения *связной целостности*.

При поддержании целостности взаимосвязанных данных не допускается наличия записи в подчиненной таблице, если в главной таблице отсутствует связанная с ней запись. Соответственно при первоначальной загрузке базы данных, а также корректировке, добавлении и удалении записей система допускает выполнение операции только в том случае, если она не приводит к нарушению целостности.

Связи, определенные в схеме данных, автоматически используются для объединения таблиц при разработке многотабличных форм, запросов, отчетов, существенно упрощая процесс их конструирования.

В схеме данных связи могут устанавливаться для любой пары таблиц, имеющих одинаковое поле, позволяющее объединять эти таблицы.

Объекты MS Access

База данных MS Access включает следующие объекты:

- таблицы;
- запросы;
- формы;
- отчеты;
- макросы;
- модули.

Таблицы создаются пользователем для организации и хранения данных. Каждая таблица хранит данные экземпляров одной сущности – одном информационном объекте модели данных предметной области. Таблица состоит из полей (столбцов) и записей (строк). В каждой записи собраны сведения об одном экземпляре информационного объекта (сущности).

Запросы на выборку служат для выборки нужных данных из одной или нескольких связанных таблиц. Результатом выполнения запроса является виртуальная таблица, которая может быть использована наряду с другими таблицами базы при обработке данных. Запрос может формироваться с помощью конструктора запросов или инструкции языка SQL. Запросы на изменение позволяют обновлять, удалять или добавлять данные в таблицы, а также создавать новые таблицы на основе существующих.

Формы являются основным средством создания диалогового интерфейса приложения пользователя. Форма используется для разработки интерфейса по

управлению приложением. Включаемые в форму процедуры обработки событий позволяют управлять процессом обработки данных в приложении. Такие процедуры хранятся в модуле формы. В формы могут вставляться рисунки, диаграммы, звуковые фрагменты, видео. Возможна разработка форм с набором вкладок, с каждой из которых связано выполнение той или иной функции приложения.

Отчеты предназначены для формирования на основе данных базы выходных документов любых форматов, содержащих результаты решения задач пользователя и вывода их на печать. Как и формы, отчеты могут включать процедуры обработки событий. Использование графических объектов позволяет дополнять данные отчета иллюстрациями. Отчеты обеспечивают возможность анализа данных при использовании фильтрации, агрегирования и представления данных источника в различных разрезах.

Макросы являются программами, состоящими из последовательности макрокоманд, которая выполняется по вызову или при наступлении некоторого события в объекте приложения или его элементе управления. Макросы позволяют автоматизировать некоторые действия в приложении пользователя. Создание макросов осуществляется в диалоговом режиме путем выбора нужных макрокоманд и задания параметров, используемых ими при выполнении.

Модули содержат процедуры на языке Visual Basic for Applications. Могут создаваться процедуры-подпрограммы, процедуры-функции, которые разрабатываются пользователем для реализации нестандартных функций в приложении пользователя, и процедуры для обработки событий. Использование процедур позволяет создать законченное приложение, которое имеет собственный графический интерфейс пользователя, позволяющий запросить выполнение всех функций приложения, обработать все ошибки и нестандартные ситуации [9].

2.4.2 MySQL

Благодаря тому, что MySQL является одной из немногих СУБД с открытым исходным кодом, она получила широкое распространение во всем мире и является одной из самых популярных СУБД, используемых в Интернете веб-разработчиками и хост-провайдерами. СУБД MySQL, впервые разработанная в 1995 г. компанией MySQL AB, на сегодняшний день принадлежит одной из крупнейших корпораций в области разработки программного обеспечения

ORACLE. СУБД MySQL успешно работает под управлением свыше 20 различных операционных систем (Windows, Linux, OS/2, Mac OS X, FreeBSD, Solaris, SunOS, OpenBSD, NetBSD, AIX, DEC UNIX, Tru64 UNIX и т. д.), а также предоставляет интерфейс для взаимодействия со множеством языков программирования: C, C++, Eiffel, Java, Perl, PHP, Python, Ruby и Tel.

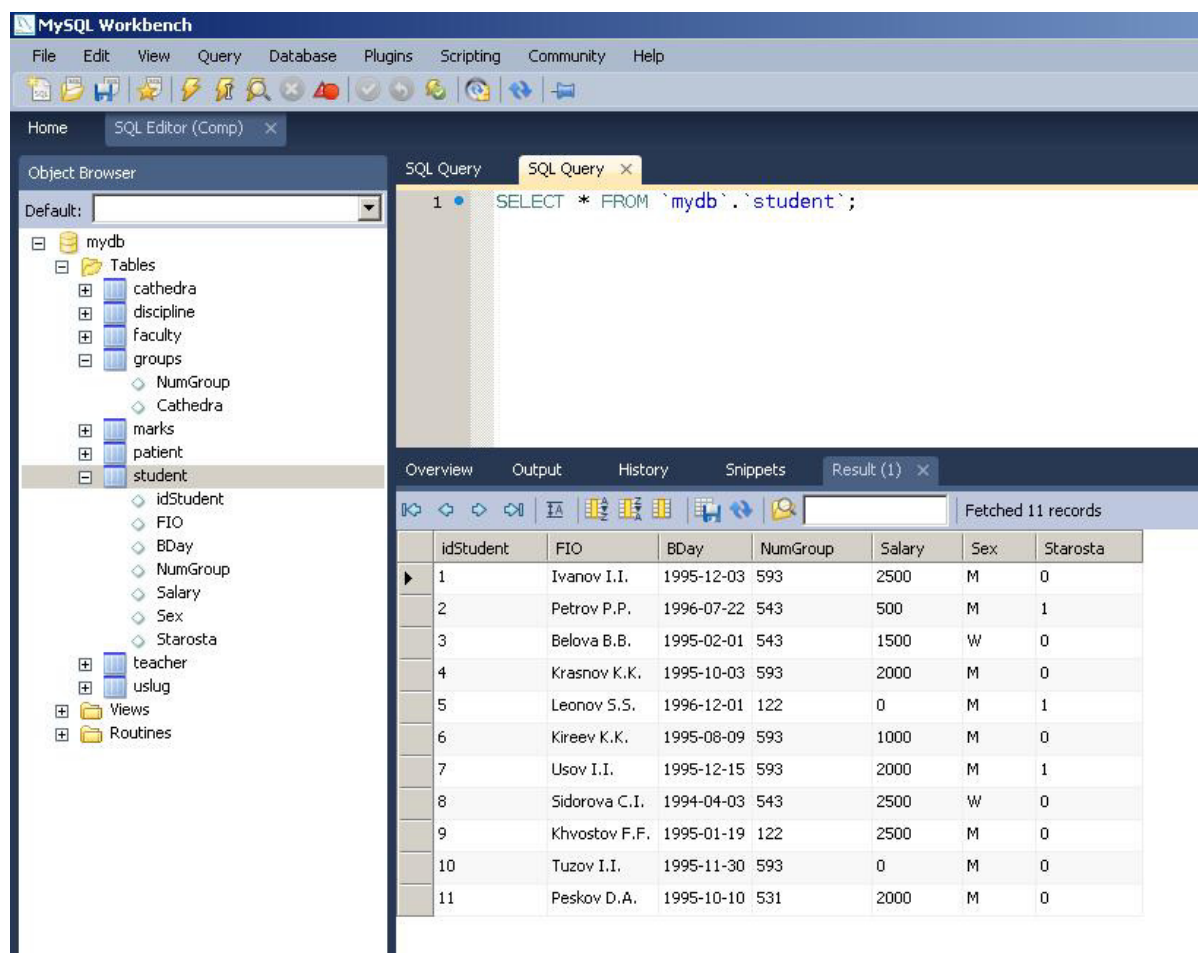


Рис. 2.8 – MySQL Workbench 5.2

MySQL является классической клиент-серверной СУБД и чаще всего используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы. В отличие от MS Access использование MySQL предполагает определенный уровень знаний и навыков у программиста баз данных, кроме того, MySQL, как и большинство СУБД, не содержит в себе средств создания пользовательских приложений. Разработчику при использовании данной СУБД в обязательном порядке необходимо предусмотреть те программные среды и дополнительные средства визуализации и настройки MySQL, которые он будет использовать для взаимо-

действия с базами данных. Например, программа MySQL Workbench, представленная на рисунке 2.8, предоставляет возможность управления СУБД MySQL с помощью удобного графического интерфейса. MySQL Workbench – инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое комплексное окружение для системы баз данных MySQL.

К основным достоинствам MySQL относятся следующие:

- скорость выполнения запросов. Наряду с Oracle, MySQL считается одной из самых быстрых СУБД в мире;
- СУБД MySQL разработана с использованием языков C/C++ и оттестирована более чем на 23 платформах;
- открытый код, который доступен для просмотра и модернизации всем желающим. Лицензия GPL (General Public License, общедоступная лицензия) позволяет постоянно улучшать программный продукт и быстро находить и устранять уязвимые места. Особенностью лицензии GPL является тот факт, что любой код, скомпилированный с GPL-кодом, попадает под GPL-лицензию, т. е. может свободно распространяться, и условием его распространения является предоставление исходных кодов;
- высокое качество СУБД MySQL;
- СУБД MySQL поддерживает API (Application Programming Interface, программный интерфейс приложения) для C, C++, Eiffel, Java, Perl, PHP, Python, Ruby и Tel. MySQL можно успешно применять как для построения веб-страниц с использованием Perl, PHP и Java, так и для работы прикладной программы, созданной с использованием Builder C++ или платформы .NET;
- наличие встроенного сервера. СУБД MySQL может быть использована как с внешним сервером, поддерживающим соединение с локальной машиной и с удаленным хостом, так и в качестве встроенного сервера. Достаточно скомпилировать программу с библиотекой встроенного сервера, и приложение будет содержать в себе полноценную СУБД MySQL с возможностью создания баз данных, таблиц и осуществления запросов к ним;
- широкий выбор типов таблиц, в том числе и сторонних разработчиков, что позволяет реализовать оптимальную для решаемой задачи производительность и функциональность. Разработчик имеет возмож-

ность выбрать наиболее быстрый тип таблиц MyISAM либо более сложный вариант InnoDB с поддержкой объемов информации до 1 Тбайт с выполнением транзакций на уровне строк;

- локализация в MySQL выполнена корректно и обеспечивает поддержку базой данных сортировки русских строк;
- MySQL, начиная с версии 5.0, практически полностью удовлетворяет стандарту SQL и, следовательно, совместима с другими базами данных [10].

На официальном сайте MySQL (<http://www.mysql.com>) всегда доступна для скачивания последняя рабочая версия сервера данной СУБД.



Контрольные вопросы по главе 2

1. Дайте определение понятиям «сущность» и «отношение», поясните как связаны эти термины между собой.
2. В чем отличия между сущностью и экземпляром сущности?
3. Перечислите свойства таблицы, являющейся отношением.
4. Назовите основные функции первичного ключа.
5. Какой вид связи в БД является самым распространенным?
6. Назовите основные объекты реляционной СУБД MS Access.

3 Язык SQL

SQL (Structured Query Language – структурированный язык запросов) – язык запросов, используемый при работе с реляционными базами данных. Термин «запрос» означает произведение операции выборки данных или операции изменения данных, таких как добавление, изменение или удаление. С самого начала существования SQL стал полным языком баз данных, т. е. его конструкции позволяют выполнять весь набор операций с базой данных: создание произвольных объектов БД (таблиц, представлений, индексов и т. п.), изменение структуры объектов БД, удаление объектов и т. д.

3.1 История SQL

Язык SQL появился после создания реляционной алгебры в середине 1970-х гг., он был разработан фирмой IBM в рамках проекта экспериментальной реляционной СУБД System R. Язык SQL, благодаря своей простоте и удобству, получил широкое распространение и постепенно стал фактическим стандартом для языков управления данными в реляционных СУБД. Большинство реляционных СУБД в той или иной степени реализовали возможности использования языка SQL, что привело к универсальности при работе с произвольными СУБД и позволило разработать единые юридические стандарты SQL. Такого рода стандарты разрабатываются специальными международными организациями по стандартизации. Каждый такой стандарт представляет собой объемный документ, тщательным образом описывающий все команды и функции языка.

Первый международный стандарт SQL (SQL1) был принят в 1989 г. Американским национальным институтом стандартов ANSI как ANSI X3.135-1989 или ANSI SQL/89. Данный стандарт в дальнейшем был также одобрен Международной организацией стандартов (ISO – International Standards Organization) в документе ISO 9075:1989. Дальнейшее развитие информационных технологий, связанных с базами данных, потребовало расширения и доработки первого стандарта SQL. Так, в конце 1992 г. был принят новый международный стандарт языка SQL – SQL/92 или SQL2, а в 1999 г. – SQL3.

Нужно заметить, что ни одна СУБД не реализует стандарт SQL в полном объеме. Кроме того, во всех расширениях языка, поддерживаемых разными СУБД, имеются возможности, не являющиеся стандартными. Таким образом,

можно сказать, что каждое расширение – это надмножество некоторого подмножества стандарта SQL.

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например вместо «отношений» используются «таблицы», вместо «кортежей» – «строки», вместо «атрибутов» – «колонки» или «столбцы». Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

SQL нельзя в полной мере отнести к традиционным языкам программирования, т. к. он не содержит, например, операторы, управляющие ходом исполнения программы, операторы описания типов и т. д. Язык SQL содержит лишь набор стандартных операторов доступа и управления к данным в БД. Операторы SQL с помощью специальных средств можно встроить в программное обеспечение, использующее базовый язык программирования, например C++ [8].

3.2 Основные операторы языка SQL

Все операторы языка SQL можно условно разделить на несколько подклассов языка [11–12]:

1. DDL – Data Definition Language (язык описания или определения данных).
2. DML – Data Manipulation Language (язык манипулирования данными).
3. DQL – Data Query Language (язык запросов к данным).
4. TCL – Transaction Control Language (язык управления транзакциями).
5. AL – Administration Language (язык администрирования).



.....

*Основным и самым используемым оператором языка SQL является оператор **SELECT** из класса DQL, который позволяет извлекать данные из одной или нескольких таблиц.*

.....

Несмотря на то что данный оператор позволяет производить различные манипуляции с данными, например складывать данные разных столбцов или производить другие вычисления внутри запроса, фактически сами данные, располагаемые в таблицах, остаются нетронутыми. Таким образом, оператор как бы извлекает актуальную копию данных из таблицы и передаёт её для дальнейшей обработки.

Класс DML, в отличие от DQL, наоборот, предназначен для произведения операций изменения имеющегося набора данных, таких как добавление, обновление или удаление.

Операторы групп DDL и AL предназначены для управления самими объектами баз данных. TCL команды работают с транзакциями, состоящими из набора команд DQL и DML. Основные операторы языка SQL приведены в таблице 3.1.

Таблица 3.1 – Основные операторы языка SQL

Класс	Оператор	Применение
DQL	SELECT	Используется для извлечения данных из базы данных
DML	INSERT UPDATE DELETE	Команды, предназначенные для ввода новых, изменения существующих и удаления ненужных строк из таблиц базы данных
DDL	CREATE ALTER DROP RENAME TRUNCATE	Используются для создания, изменения и удаления объектов базы данных (таблиц, представлений, последовательностей и т. п.)
TCL	COMMIT ROLLBACK SAVEPOINT SET TRANSACTION	Команды управления транзакциями. Управляют изменениями, которые производятся с помощью команд DML
AL	GRANT REVOKE	Команды управления доступом к объектам

Данный набор команд не является полным, ведь существуют также специализированные операторы языка SQL, которые позволяют работать, например, с курсорами или управлять динамическим запуском других операторов. Также расширения языка SQL в различных СУБД содержат дополнительные операторы, позволяющие осуществлять более гибкое и функциональное управление данными в БД.

3.3 Типы данных

В языке SQL, так же как и в различных языках программирования, для хранения данных поддерживается набор типов данных. Типы данных отлича-

ются друг от друга в организации хранения данных и применяемым к ним операциям. Очевидно, что операции деления или умножения относятся к числовым типам данных и не применимы к символьным (строкам). Таким образом, некоторый набор операций может быть применим к различным типам данных, а некоторые операции могут применяться только для отдельного типа. Например, операция сложения «+» может быть применима как к числовым данным ($4 + 7 = 11$), так и к текстовым («четыре » + «семь» = «четыре семь»).

В различных СУБД набор поддерживаемых типов данных для столбцов таблиц отличается друг о друга. В стандартах SQL, в зависимости от времени их появления, также можно найти некоторые отличия в представленных типах данных. Тем не менее, существует некоторый базовый набор, который будет поддерживаться большинством стандартов и СУБД.

Типы данных SQL:

1. Числовой (целочисленный):
 - INTEGER;
 - SMALLINT;
 - BIGINT.
2. Числовой (вещественный или дробный):
 - REAL;
 - FLOAT;
 - NUMERIC;
 - DECIMAL.
3. Символьный (строковый):
 - CHAR;
 - VARCHAR;
 - TEXT.
4. Дата (время):
 - DATE;
 - TIME.
5. Логический:
 - BOOLEAN.

Представленные типы данных являются наиболее часто употребляемыми при создании столбцов таблиц и позволяют организовать рациональное хранение информации и осуществлять соответствующие операции над данными.

3.4 Оператор SELECT

Для выборки строк из таблиц базы данных используется команда SELECT.

Синтаксис команды SELECT выглядит следующим образом:

```
SELECT [ DISTINCT ] { * | столбец [ псевдоним ], ... }
FROM таблица
[WHERE условие]
[ORDER BY {столбец | выражение [ASC | DESC], ... }];
```

Рассмотрим подробнее каждый из пунктов данного синтаксиса. Ключевых операторов здесь только два: SELECT и FROM. Любой запрос на выборку данных из таблиц должен состоять как минимум из 2 данных операторов.

После оператора SELECT обязательно необходимо указать имена столбцов таблицы, которые должны отображаться в запросе. Если необходимо использовать все столбцы таблицы в запросе, пользователь вместо их последовательного перечисления может использовать символ *. После оператора FROM обязательно необходимо указать имя таблицы или нескольких таблиц, данные из которой будут отображены в итоге.

Например, в результате запуска команды `SELECT * FROM Студент` на экран будут выведены все строки из таблицы «Студент».

Необязательным, но довольно часто используемым является оператор WHERE, который используется для отбора строк, соответствующих некоторому условию. Условие представляет собой некоторое логическое выражение, которое может состоять из имен столбцов, выражений, констант, операторов сравнения и логических операторов. В результирующую выборку попадут только те строки таблицы, для которых заданное логическое выражение будет иметь значение 'истина'.

Последний оператор ORDER BY используется для сортировки данных выборки по одному или нескольким полям в порядке возрастания или убывания.

Оператор SELECT используется именно для выборки необходимых данных. Получив эту команду от приложения-клиента, приложение-сервер отбирает столбцы и строки, удовлетворяющие условиям запроса, и передает их обратно клиенту. То, что произойдет с полученными данными дальше, зависит от функций приложения-клиента. Данные могут быть просто выведены на экран,

на печать, сохранены в файл, переданы на обработку и т. п. За все эти операции команда `SELECT` уже не отвечает.

Рассмотрим примеры различных запросов к отношению «Студент», приведённому в таблице 3.2.

Таблица 3.2 – Отношение «Студент»

Номер	ФИО	Дата рождения	Группа	Стипендия, руб.
1	Иванов Сергей Петрович	14.05.1988	116	1500
2	Петров Иван Сергеевич	27.01.1989	598	500
3	Алексеев Семен Олегович	05.10.1993	445	1500
4	Белов Петр Иванович	01.11.1988	116	2500
5	Сидорова Анна Игоревна	05.04.1991	116	1000
6	Егорова Ирина Петровна	18.12.1992	445	–
7	Широков Иван Дмитриевич	22.07.1991	445	2000



Пример 3.1

Вывод списка фамилий, имен, отчеств (ФИО) и номеров групп студентов, реализуемый с помощью запроса `SELECT ФИО, Группа FROM Студент`, приведен в таблице 3.3.

Таблица 3.3 – Список фамилий и номеров групп

ФИО	Группа
Иванов Сергей Петрович	116
Петров Иван Сергеевич	598
Алексеев Семен Олегович	445
Белов Петр Иванович	116
Сидорова Анна Игоревна	116
Егорова Ирина Петровна	445
Широков Иван Дмитриевич	445

После оператора `SELECT` можно использовать не только имена столбцов, но и выражения, составленные с помощью арифметических операторов, имен столбцов и констант.



Пример 3.2

Вывод списка студентов и размера их стипендий за год, реализуемый с помощью запроса `SELECT ФИО, Стипендия*12 as "Стипендия за год" FROM Студент`, приведен в таблице 3.4.

Таблица 3.4 – Список фамилий и стипендий за год

ФИО	Стипендия за год, руб.
Иванов Сергей Петрович	18 000
Петров Иван Сергеевич	6 000
Алексеев Семен Олегович	18 000
Белов Петр Иванович	30 000
Сидорова Анна Игоревна	12 000
Егорова Ирина Петровна	—
Широков Иван Дмитриевич	24 000

Используя ключевое слово `AS`, выражению в `SELECT` можно присвоить псевдоним. Псевдоним используется в качестве имени столбца в данном запросе.

В предложении `WHERE` могут использоваться следующие операторы сравнения:

1. Простые операторы сравнения:

- `=` равно;
- `<>` не равно;
- `>` больше;
- `>=` больше или равно;
- `<` меньше;
- `<=` меньше или равно.

2. Логические операторы сравнения:

- AND – логический оператор И. Он используется, когда необходимо, чтобы в выборку попали записи, для которых одновременно выполняются два условия.
- OR – логический оператор ИЛИ. Он используется, когда необходимо, чтобы в выборку попали записи, для которых выполняется как минимум одно из двух условий.
- NOT – логический оператор НЕ. Он используется для того, чтобы инвертировать результат какого-либо условия.

3. Специальные операторы сравнения SQL:

- BETWEEN...AND... – используется для поиска значений попадающих в заданный интервал.
- LIKE – позволяет производить поиск по символьному шаблону.
- IN (список) – используется для поиска значений, совпадающих с каким-либо значением из списка.
- IS NULL – используется для поиска пустых значений.



Пример 3.3

Вывод списка студентов из группы 116 с размером стипендии более 1000, реализуемый с помощью запроса `SELECT * FROM Студент WHERE Группа=116 AND Стипендия>1000`, приведен в таблице 3.5.

Таблица 3.5 – Список студентов группы 116 и стипендией более 1000 руб.

Номер	ФИО	Дата рождения	Группа	Стипендия, руб.
1	Иванов Сергей Петрович	14.05.1988	116	1500
4	Белов Петр Иванович	01.11.1988	116	2500

В данном примере использовалась комбинация простых операторов = и > и логического оператора AND.



Пример 3.4

Вывод ФИО студентов из групп 116 и 598, который можно реализовать с помощью запроса `SELECT ФИО FROM Студент WHERE Группа IN (116, 598)`, приведен в таблице 3.6.

Таблица 3.6 – Список ФИО из групп 116, 598

ФИО
Иванов Сергей Петрович
Петров Иван Сергеевич
Белов Петр Иванович
Сидорова Анна Игоревна

В данном примере использовался специальный оператор `IN`, хотя во многих ситуациях он может быть заменён комбинацией простых и логических операторов. Например, результатом запроса `SELECT ФИО FROM Студент WHERE Группа=116 OR Группа=598` будет также таблица 3.6.



Пример 3.5

Вывод отсортированного по номеру группы списка фамилий и номеров групп студентов, реализуемый с помощью запроса `SELECT ФИО, Группа FROM Студент ORDER BY Группа ASC`, приведен в таблице 3.7.

Таблица 3.7 – Отсортированный список ФИО и номеров групп

ФИО	Группа
Иванов Сергей Петрович	116
Белов Петр Иванович	116
Сидорова Анна Игоревна	116
Алексеев Семен Олегович	445
Егорова Ирина Петровна	445
Широков Иван Дмитриевич	445
Петров Иван Сергеевич	598

Сортировать результат можно также по нескольким столбцам. Столбцы указываются в предложении ORDER BY через запятые. Если необходимо сортировать данные по возрастанию, то после названия столбца указывается оператор ASC или ничего не указывается. В случае, когда сортировка идет по убыванию, в обязательном порядке необходимо задать ключевое слово DESC после имени столбца.

.....

3.5 Операторы DML

3.5.1 INSERT

Для вставки строк в таблицу базы данных используется команда INSERT.

Синтаксис команды INSERT выглядит следующим образом:

```
INSERT INTO таблица
[ (НазваниеСтолбца1, НазваниеСтолбца2, ...) ]
VALUES (Значение1, Значение2, ...);
```

Ключевыми операторами здесь являются INSERT INTO и VALUES. После первого из них необходимо указать таблицу, в которую будут добавлены данные, а после оператора VALUES в скобках через запятую указываются данные, которые необходимо добавить.



Пример 3.6

.....

Добавление в таблицу «Студент» новой записи будет реализовано с помощью запроса INSERT INTO Студент (Номер, ФИО, «Дата рождения», Группа, стипендия) values (1, «Тимофеев Н.И.», «23.05.1987», 116, 2000);.

В том случае, если в запросе на добавление данных участвуют все поля, то список полей можно не размещать в запросе. В этом случае он примет вид: INSERT INTO Студент values (1, «Тимофеев Николай Игоревич», «23.05.1987», 116, 2000);.

.....

3.5.2 UPDATE

Для изменения значений в строках таблиц базы данных используется команда UPDATE.

Синтаксис команды UPDATE выглядит следующим образом:

```
UPDATE таблица
SET НазваниеСтолбцаN = ВыражениеN
[, НазваниеСтолбцаM = ВыражениеM, ...]
[WHERE условие];
```

Ключевых оператора здесь два: UPDATE и SET. После оператора UPDATE указывается таблица, в которой необходимо произвести изменения, а после оператора SET указываются непосредственно операции по изменению данных. Оператор WHERE является необязательным, т. е. при его отсутствии команда UPDATE произведет изменения во всех записях таблицы, а при его наличии – только в тех, которые указаны в условии.



Пример 3.7

Необходимо назначить стипендию 1 500 руб. студенту Петрову Ивану Сергеевичу из группы 598. Подобный запрос будет выглядеть следующим образом:

```
UPDATE Студент SET Стипендия=1500 WHERE ФИО = "Петров
Иван Сергеевич" AND Группа = 598;
```



Пример 3.8

Необходимо увеличить на 30% стипендию всем студентам. Подобный запрос будет выглядеть следующим образом:

```
UPDATE Студент SET Стипендия = Стипендия*1,3;
```

3.5.3 DELETE

Для удаления строк из таблиц базы данных используется команда DELETE.

Синтаксис команды DELETE выглядит следующим образом:

```
DELETE FROM таблица
[WHERE условие];
```

Ключевых оператора здесь только два: DELETE и FROM. Любой запрос на выборку данных из таблиц должен состоять как минимум из 2 данных операторов.

ров. Оператор WHERE является необязательным, т. е. при его отсутствии команда DELETE произведет удаление всех записей таблицы, а при его наличии – только в тех, которые указаны в условии.



Пример 3.9

Удалить из таблицы записи о студентах группы 598. Подобный запрос будет выглядеть следующим образом:

```
DELETE FROM Студент WHERE Группа = 598;
```



Пример 3.10

Удалить из таблицы запись о студенте Петрове Иване Сергеевиче. Подобный запрос будет выглядеть следующим образом:

```
DELETE FROM Студент WHERE ФИО = "Петров Иван Сергеевич";
```

3.6 Групповые функции

Для работы с группой записей и подсчета различных итоговых результатов для групп в языке SQL существуют групповые функции. Принцип работы групповой функции состоит в выведении одного результирующего значения для группы строк. Групповые функции используются вместе с оператором SELECT, который в таком случае имеет следующий синтаксис:

```
SELECT [столбец|выражение, ] групповая_функция
FROM таблица1, ...
[WHERE условие]
[GROUP BY выражение_группирования]
[HAVING условие_включения_группы]
[ORDER BY {столбец | выражение [ASC | DESC], ... }];
```

По сравнению с простым запросом групповой запрос обязательно содержит групповую функцию, а также может содержать операторы GROUP BY и HAVING. Оператор GROUP BY позволяет весь диапазон строк разбить на группы согласно выражению группирования, в результате чего групповая функция будет производить расчет результата для каждой такой группы. В случае, когда

оператор `GROUP BY` отсутствует, групповая функция работает для всего набора строк из таблицы. Оператор `HAVING` позволяет производить фильтрацию итоговых строк согласно некоторому условию включения группы. В таблице 3.8 приведены наиболее часто используемые групповые функции.

Таблица 3.8 – Групповые функции языка SQL

Функция	Описание
<code>AVG ()</code>	Вычисляет среднее арифметическое набора числовых значений поля запроса по группе записей
<code>COUNT ()</code>	Вычисляет количество записей в группе
<code>MAX ()</code>	Возвращает максимальное значение указанного поля из группы записей
<code>MIN ()</code>	Возвращает минимальное значение указанного поля из группы записей
<code>SUM ()</code>	Вычисляет сумму значений указанного поля по группе записей

Функции `AVG` и `SUM` могут применяться только к столбцам с числовыми данными, а функции `MAX`, `MIN` и `COUNT` – к столбцам любого типа.



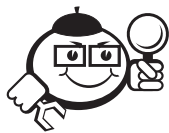
Пример 3.11

Расчет суммарной стипендии всех студентов из таблицы «Студент», реализуемый с помощью запроса `SELECT SUM(Стипендия) AS 'Суммарная стипендия' FROM Студент`, приведен в таблице 3.9.

Таблица 3.9 – Суммарная стипендия всех студентов

Суммарная стипендия
9000

В данном примере не была использована группировка, а потому групповая функция произвела подсчет для всех строк.



Пример 3.12

Расчет суммарной стипендии в каждой группе из таблицы «Студент», реализуемый с помощью запроса `SELECT SUM(Стипендия) AS 'Суммарная стипендия' FROM Студент GROUP BY Группа`, приведен в таблице 3.10.

Таблица 3.10 – Суммарная стипендия в каждой группе

Группа	Суммарная стипендия, руб.
116	5000
598	500
446	3500

В данном примере с помощью группировки все строки были разбиты на 3 группы строк, в которых у каждой записи было одинаковым значение поля «Группа», и групповая функция произвела подсчет суммы для каждой из этих групп.



Пример 3.13

Подсчёт количества студентов, родившихся после 01.01.1990 г., реализуемый с помощью запроса `SELECT COUNT(ФИО) AS 'Количество студентов' FROM Студент WHERE «Дата рождения» > «01.01.1990»`, приведен в таблице 3.11.

Таблица 3.11 – Количество студентов, родившихся после 1990 г.

Количество студентов
4

В данном примере функция `COUNT` производит простой подсчет количества строк, удовлетворяющих условию. В качестве аргумента данной функции можно использовать произвольный атрибут.



Пример 3.14

Расчет суммарной стипендии в каждой группе из таблицы «Студент» и вывод списка только тех групп, в которых суммарная стипендия больше 1 000 руб., реализуемый с помощью запроса `SELECT SUM(Стипендия) FROM Студент GROUP BY Группа HAVING SUM(Стипендия) > 1000`, приведен в таблице 3.12.

Таблица 3.12 – Суммарная стипендия в каждой группе

Группа	SUM (Стипендия, руб.)
116	5000
446	3500

В данном примере с помощью оператора `HAVING` происходит фильтрация результатов группировки и оператор `WHERE` в таком случае использовать нельзя, т. к. он работает только с исходным набором строк, но не с полученным после группировки.

3.7 Запросы к нескольким таблицам

В большинстве баз данных информация логически распределена по многим таблицам и для выдачи необходимой пользователю информации данные приходится запрашивать у двух или более таблиц. В качестве примера рассмотрим фрагмент БД «Студенты», который состоит из двух связанных таблиц «Группа» и «Студент» (рис. 3.1).

Синтаксис команды `SELECT` позволяет после оператора `FROM` использовать не одну, а целый набор таблиц, что и позволяет в дальнейшем извлекать необходимую информацию, логически размещенную в разных таблицах. В тех ситуациях, когда в запросе используется более одной таблицы, для однозначной идентификации названия полей после команды `SELECT` принято вместо простого названия поля таблицы указывать сначала имя таблицы, а затем, после точки, — имя поля.

«Группа»

Группа	Специальность	Факультет
116	Электронная техника	ЭТ
445	Менеджмент	Эконом.
598	Информационные системы	ВС

«Студент»

Номер	ФИО	Группа	Стипендия
1	Иванов Сергей Петрович	116	1500
2	Петров Иван Сергеевич	598	500
3	Алексеев Семен Олегович	445	1500
4	Белов Петр Иванович	116	2500

Рис. 3.1 – Фрагмент БД «Студенты»

Например, вместо `SELECT ФИО, Стипендия FROM Студент, Группа;` запрос пишется в виде строки `SELECT Студент.ФИО, Студент.Стипендия FROM Студент, Группа.`

Это связано, прежде всего с тем, что в различных таблицах поля могут иметь одинаковые названия, например, в используемых таблицах нашего примера поле «Группа» имеется в обеих таблицах. Если поле имеет уникальное имя во всей базе данных, то процессор СУБД позволяет не указывать имя таблицы перед именем поля. Таким образом, в приведенном выше примере допустимыми являются оба варианта запросов, однако в данном разделе для удобства восприятия информации будет использован синтаксис с полным указанием таблиц и полей.

3.7.1 Декартово произведение записей таблиц

Самым простым с точки зрения синтаксиса запросом на вывод информации из нескольких таблиц является так называемое *декартово произведение таблиц*. Для нашего примера такой запрос будет выглядеть следующим образом: `SELECT Студент.*, Группа.* FROM Студент, Группа.` Данный запрос выведет все поля и записи из двух таблиц сразу, при этом «умножив» все записи друг на друга. В результате пользователь получит результат, приве-

денный в таблице 3.13. Количество записей, выведенных в результате такого запроса, будет равно произведению количества записей первой таблицы на количество записей второй таблицы.

Таблица 3.13 – Декартово произведение таблиц

Студент. Номер	Студент. ФИО	Студент. Группа	Студент. Стипендия, руб.	Группа. Группа	Группа. Специальность	Группа. Факультет
1	Иванов Сергей Петрович	116	1500	116	Электронная техника	ЭТ
2	Петров Иван Сергеевич	598	500	116	Электронная техника	ЭТ
3	Алексеев Семен Олегович	445	1500	116	Электронная техника	ЭТ
4	Белов Петр Иванович	116	2500	116	Электронная техника	ЭТ
1	Иванов Сергей Петрович	116	1500	445	Менеджмент	Эконом.
2	Петров Иван Сергеевич	598	500	445	Менеджмент	Эконом.
3	Алексеев Семен Олегович	445	1500	445	Менеджмент	Эконом.
4	Белов Петр Иванович	116	2500	445	Менеджмент	Эконом.
1	Иванов Сергей Петрович	116	1500	598	Информационные системы	ВС
2	Петров Иван Сергеевич	598	500	598	Информационные системы	ВС
3	Алексеев Семен Олегович	445	1500	598	Информационные системы	ВС

Студент. Номер	Студент. ФИО	Студент. Группа	Студент. Стипендия, руб.	Группа. Группа	Группа. Специальность	Группа. Факультет
4	Белов Петр Иванович	116	2500	598	Информационные системы	ВС

Подобные запросы на практике используются крайне редко, поскольку фактически лишены смысла. Тем не менее, важно понимать, что в любом случае, когда в операторе `SELECT` используется более одной таблицы, всегда происходит операция декартова произведения всех записей таблиц. Если в запросе используется 3 и более таблицы, то выполнение этого запроса приведет к «умножению» первой таблицы на вторую, а затем полученная таблица будет «умножена» на третью и т. д. Смысл большинства подобных запросов приобретает благодаря использованию необходимых условий после оператора `WHERE`.

Видоизменим наш запрос, исключив из него лишние поля, и попробуем вывести список студентов, а также группы и факультеты, на которых они обучаются. Для наглядности и понимания в запросе оставим поля «Группа» из обеих таблиц.

Без применения оператора `WHERE` будет использоваться следующий запрос: `SELECT Студент.ФИО, Студент.Группа, Группа.Группа, Группа.Факультет FROM Студент, Группа.`

Результатом выполнения запроса будет таблица 3.14.

Таблица 3.14 – Запрос «Студент – Группа – Факультет»

Студент.ФИО	Студент.Группа	Группа.Группа	Группа.Факультет
Иванов Сергей Петрович	116	116	ЭТ
Петров Иван Сергеевич	598	116	ЭТ
Алексеев Семен Олегович	445	116	ЭТ
Белов Петр Иванович	116	116	ЭТ
Иванов Сергей Петрович	116	445	Эконом.
Петров Иван Сергеевич	598	445	Эконом.
Алексеев Семен Олегович	445	445	Эконом.
Белов Петр Иванович	116	445	Эконом.

Студент.ФИО	Студент.Группа	Группа.Группа	Группа.Факультет
Иванов Сергей Петрович	116	598	ВС
Петров Иван Сергеевич	598	598	ВС
Алексеев Семен Олегович	445	598	ВС
Белов Петр Иванович	116	598	ВС

Полученная в результате запроса таблица практически аналогична предыдущей и отличается от неё только отсутствием некоторых полей. Как видно, сформированный запрос не совсем корректно отображает информацию о студентах вуза, их группах и факультетах, поскольку по-прежнему просто осуществляет «умножение» всех записей. Тем не менее, среди всех полученных в результате запроса записей, имеются 4 «правильные» записи, отображающие корректную информацию о студентах. К таковым записям относятся только те, у которых значения полей «Студент.Группа» и «Группа.Группа» совпадают.

В результате можно сделать вывод, что для формирования корректного запроса на выборку из нескольких таблиц *необходимо добавить в исходный запрос условие о равенстве значений двух связанных полей таблиц*:

```
SELECT Студент.ФИО, Студент.Группа, Группа.Группа,
Группа.Факультет FROM Студент, Группа WHERE Студент.Группа=Группа.Группа;
```

Результатом выполнения подобного запроса будет таблица 3.15.

Таблица 3.15 – Запрос «Студент – Группа – Факультет» с условием

ФИО	Студент.Группа	Группа.Группа	Факультет
Иванов Сергей Петрович	116	116	ЭТ
Белов Петр Иванович	116	116	ЭТ
Алексеев Семен Олегович	445	445	Эконом.
Петров Иван Сергеевич	598	598	ВС

В подобных запросах обычно не выводят значение двух связанных полей, как в данном примере, однако пользователю всегда необходимо указывать, из какой таблицы он будет выводить значение связанного поля.

В итоге можно сформулировать правильный синтаксис оператора SELECT для вывода связанных записей из двух и более таблиц.

```
SELECT Список_полей из Таблицы1, Таблицы2, .. , Таб-
лицыN
FROM Таблица1, Таблица2, .. , ТаблицаN
WHERE Таблица1.ПолеСвязи1=Таблица2.ПолеСвязи2 AND ...
Таблица2.ПолеСвязи3=ТаблицаN.ПолеСвязиM;
```

3.7.2 Соединение таблиц

Операции соединения таблиц возвращают пользователю набор записей, являющийся некоторой комбинацией записей соединяемых таблиц и обязательно содержащий оператор JOIN (от англ. «соединить»).

Существует целый набор разновидностей соединения таблиц, реализовать который помогают специальные операторы, добавляемые к ключевому оператору JOIN.

Базовый синтаксис оператора выглядит следующим образом:

```
SELECT Список_полей
FROM Таблица1 JOIN Таблица2 ON Табли-
ца1.ПолеСвязи1=Таблица2.ПолеСвязи2;
```

Вместо ключевого слова JOIN можно применять следующие комбинации операторов:

- INNER JOIN (или просто JOIN);
- LEFT OUTER JOIN (LEFT JOIN);
- RIGHT OUTER JOIN (RIGHT JOIN);
- FULL OUTER JOIN (FULL JOIN);
- CROSS JOIN.

Самым используемым из приведенного списка оператором является INNER JOIN, который чаще всего используется без ключевого слова INNER. Фактически этот оператор является аналогом использования оператора SELECT для связи двух и более таблиц, приведенного выше. Оператор INNER JOIN осуществляет вывод только тех строк в итоговый запрос, данные для которых есть в обеих связываемых таблицах. То есть для каждой записи из первой таблицы обязательно должна быть связанная запись из второй, и наоборот. Строки, для которых связываемые записи отсутствуют, в итоговый запрос с оператором INNER JOIN не попадут.

Запишем предложенный выше запрос о выводе информации о студентах с помощью оператора INNER JOIN:

```
SELECT Студент.ФИО, Студент.Группа, Группа.Факультет
FROM Студент INNER JOIN Группа ON Студент.Группа=
=Группа.Группа;
```

Для примера рассмотрим работу с таблицей «Успеваемость» (табл. 3.16), которая связана с таблицей «Студент» по полю «Номер Студента».

Таблица 3.16 – Таблица «Успеваемость»

Номер Студента (ПК)	Предмет (ПК)	Дата сдачи экзамена	Оценка
1	Базы данных	14.06.2016	4
2	Базы данных	14.06.2016	5
1	Информатика	18.06.2016	4
2	Информатика	18.06.2016	4

Осуществим вывод информации из двух таблиц «Студент» и «Успеваемость», которая будет содержать поля «ФИО», «Предмет» и «Оценка».

```
SELECT Студент.ФИО, Успеваемость.Предмет, Успевае-
мость.Оценка FROM Студент JOIN Успеваемость ON Сту-
дент.Номер=Успеваемость.НомерСтудента;
```

Результат выполнения подобного запроса – таблица 3.17.

Таблица 3.17 – Запрос «Успеваемость»

ФИО	Предмет	Оценка
Иванов Сергей Петрович	Базы данных	4
Белов Петр Иванович	Базы данных	5
Иванов Сергей Петрович	Информатика	4
Белов Петр Иванович	Информатика	4

Вместо оператора INNER можно также использовать операторы LEFT или RIGHT, соответственно левое или правое соединение. Использование LEFT JOIN означает, что в результате выполнения запроса будут выведены все записи из первой (левой) таблицы и те записи из второй (правой) таблицы, для которых есть связанные записи в первой. Если для записей из первой таблицы не

окажется связанных записей во второй, то будут просто выведены пустые значения (NULL).

Для предыдущего примера изменим формулировку запроса – выведем информацию об оценках студентов, а также выведем информацию о студентах, которые ещё не сдавали экзамены.

Для реализации такого запроса в предыдущем примере необходимо всего лишь использовать LEFT JOIN вместо JOIN.

```
SELECT Студент.ФИО, Успеваемость.Предмет, Успеваемость.Оценка FROM Студент LEFT JOIN Успеваемость ON Студент.Номер=Успеваемость.НомерСтудента;
```

Результат выполнения подобного запроса – таблица 3.18.

Таблица 3.18 – Запрос «Успеваемость» с измененной формулировкой

ФИО	Предмет	Оценка
Иванов Сергей Петрович	Базы данных	4
Белов Петр Иванович	Базы данных	5
Иванов Сергей Петрович	Информатика	4
Белов Петр Иванович	Информатика	4
Алексеев Семен Олегович		
Петров Иван Сергеевич		

В данном случае были выведены все записи из таблицы «Студент» и связанные с ними записи таблицы «Успеваемость». Как видно из таблицы, для записей, у которых не было связей, были выведены пустые поля (в зависимости от используемой СУБД на экране в таких ситуациях появляются либо пустые ячейки, либо значения NULL).

Оператор RIGHT JOIN работает аналогично LEFT JOIN с той лишь разницей, что «главной» становится таблица, указанная справа от оператора.

Оператор FULL JOIN означает полное внешнее соединение и фактически является объединением левого и правого соединения. В итоговый результат попадают все строки из первой (левой) таблицы, которые дополняются пустыми значениями для несвязанных записей, а также все строки правой таблицы, которые дополняются пустыми значениями для несвязанных записей.

Оператор CROSS JOIN осуществляет вывод декартова произведения записей нескольких таблиц.

3.7.3 Объединение таблиц

Ещё одним оператором, позволяющим выводить данные нескольких таблиц, является оператор `UNION`. Данный оператор производит «объединение» строк из нескольких запросов в единый массив и имеет следующий синтаксис:

```
SELECT ... FROM ...
UNION
SELECT ... FROM ...
UNION
SELECT ... FROM ...
```

В результате выполнения к строкам первого запроса фактически будут присоединены строки второго запроса и т. д. Важно понимать, что количество полей в каждом из объединяемых запросов должно быть одинаковым, кроме того, типы полей также должны соответствовать друг другу.

Для примера рассмотрим запрос, который выводит список студентов и преподавателей, а также даты их рождения.

```
SELECT ФИО, ДатаРождения FROM Студент
UNION
SELECT ФИО, ДатаРождения FROM Преподаватель;
```

Для работы с группой записей и подсчета различных итоговых результатов для групп в языке SQL существуют групповые функции. Принцип работы групповой функции состоит в выведении одного результирующего значения для группы строк.



Контрольные вопросы по главе 3

1. Перечислите основные классы языка SQL.
2. Какие два ключевых слова обязательно используются в любой операции выборки данных из таблицы?
3. Перечислите операторы DML.
4. В чем разница между операторами `ORDER BY` и `GROUP BY`?
5. Для чего может быть использован подобный запрос: `SELECT COUNT(*) FROM Таблица1`?
6. В чем отличия оператора `HAVING` от оператора `WHERE`?
7. В чем разница между операторами `INNER JOIN` и `LEFT(RIGHT) JOIN`?
8. Назовите оператор, используемый при операции объединения таблиц.

4 Проектирование баз данных

Как уже говорилось ранее, любая база данных не может существовать сама по себе, ведь в большинстве случаев база данных является главной составной частью некоторой информационной системы. Всех пользователей такой информационной системы, а следовательно и находящейся в её составе базы данных, можно условно разделить на две категории: пользователи, работающие с информацией в базе данных, и проектировщики, участвующие в создании и поддержании базы данных в рабочем состоянии.

Первая категория пользователей осуществляет непосредственно операции над данными, такие как внесение информации в базу данных, изменение существующей информации, удаление, просмотр, распечатка и т. д. Пользователи порой даже не задумываются над тем, какие объекты используются в конкретной базе данных, а просто работают с различными инструментами, которые выводят нужную информацию на экран. Например, заходя на сайт интернет-магазина или просматривая новости, пользователи не знают, в каких таблицах хранится эта информация, а просто осуществляют взаимодействие с ней с помощью предложенного проектировщиками и разработчиками пользовательского интерфейса системы.

В задачи проектировщика входят уже совершенно другие процессы. Главной его задачей является создание корректной структуры базы данных, которая позволит оптимальным образом обеспечить хранение и управление информацией. Таким образом, проектировщики основную часть времени взаимодействуют не с данными, а с объектами баз данных, такими как таблицы, запросы, связи и т. д. Даже после создания и внедрения в работу проектировщик может вносить изменения в существующие структуры баз данных: изменять таблицы, добавлять новые объекты. Ещё одной важной задачей при разработке информационной системы является разработка интерфейса пользователя, однако эта задача напрямую не относится к проектированию базы данных, а реализуется отдельно либо после разработки готовой базы данных, либо создается параллельно с процессами проектирования БД.

4.1 Жизненный цикл БД

Все процессы, связанные с созданием, проектированием и эксплуатацией баз данных, образуют так называемый *жизненный цикл баз данных*.



Жизненный цикл баз данных (ЖЦ БД) – это непрерывный процесс, начинающийся с момента принятия решения о необходимости создания базы данных и заканчивающийся в момент полного ее изъятия из эксплуатации.

Основные этапы ЖЦ приведены на рисунке 4.1.



Рис. 4.1 – Этапы жизненного цикла БД

Остановимся подробнее на каждом из этапов ЖЦ:

1. Планирование и анализ требований к информационной системе.

Планирование будущей информационной системы является первым этапом жизненного цикла, без которого остальные этапы просто не имели бы смысла. Именно здесь выясняется необходимость в создании действительно нового подхода к дальнейшей деятельности организации, связанного с автоматизацией при работе с информацией. Это может произойти в силу разных причин, например предприятие или пользователь до этого осуществляли работу с информацией в ручном режиме или просто в режиме работы с отдельными электронными документами или файлами. Кроме того, необходимость в создании новой ИС, а вследствие и базы данных, могла возникнуть по причине устаревания уже существующей ИС.

Наиболее важными результатами этого этапа являются:

- описание целей и задач будущей ИС;
- выработка общих требований заказчика к созданию ИС;

- обследование объекта автоматизации – функций и структуры организации, перечень задач, подлежащих автоматизации, исследование уже существующей на предприятии АИС и т. д.;
- ориентировочный состав технических средств и технико-экономическое обоснование всего проекта;
- исследование информационных потребностей всех будущих пользователей системы.

Для серьезных проектов этап планирования и анализа требований заканчивается составлением *технического задания*, т. е. документа, определяющего цели, требования и основные исходные данные, необходимые для разработки автоматизированной информационной системы [13].

2. Проектирование базы данных.

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информации о некоторой предметной области к формализованному описанию объектов предметной области в терминах некоторой модели, например реляционной. Выделим основные шаги данного этапа:

- системный анализ и описание объектов предметной области;
- инфологическое проектирование;
- логическое проектирование на основе выбранной СУБД;
- физическое проектирование.

Основные этапы проектирования приведены на рисунке 4.2.

Системный анализ предметной области состоит в том, что необходимо привести подробное словесное описание всех объектов и взаимосвязей между описываемыми объектами в предметной области. Кроме того, должны быть сформулированы конкретные задачи, которые будут решаться с использованием данной БД с кратким описанием алгоритмов их решения, описанием входных и выходных документов, на основе которых будет производиться ввод и вывод информации в информационную систему.

Инфологическое проектирование – проектирование инфологической модели предметной области, т. е. формализованного описания объектов предметной области в терминах некоторой семантической модели, например в терминах модели «сущность – связь», или ER-модели.

Результатом данного этапа будет получение так называемой ER-диаграммы предметной области, которая фактически является промежуточным звеном между словесным описанием ПО и схемой данных готового проекта БД.

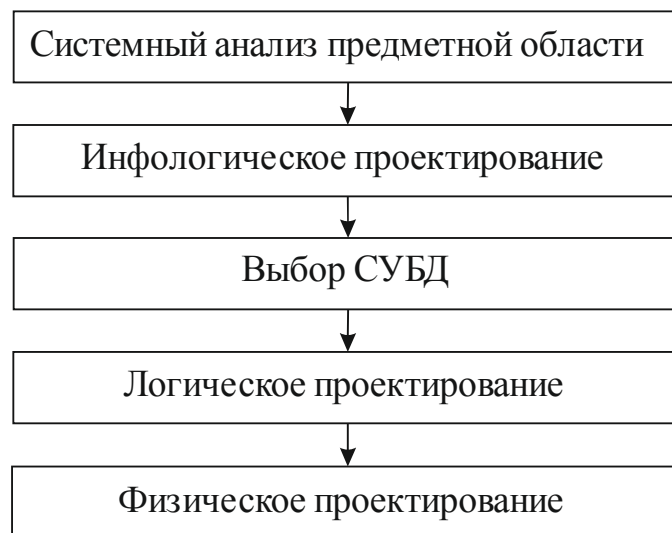


Рис. 4.2 – Этапы проектирования БД

Подробнее этот этап будет рассмотрен в п. 4.3.

Выбор и приобретение СУБД необходимо будет осуществить между вторым и третьим этапом проектирования. Стоит отметить, что инфологическое проектирование осуществляется без привязки к конкретной СУБД и в дальнейшем может быть реализовано в любой из выбранных СУБД. К факторам, которые влияют на выбор СУБД, стоит отнести следующие: количество пользователей будущей системы и частота обращений к БД, стоимость СУБД, прикладные программные средства, которые в дальнейшем будут взаимодействовать с БД, объем хранимых данных и т. д.

Логическое проектирование – процесс разработки корректной схемы базы данных в терминах выбранной СУБД. В литературе, помимо термина «логическое проектирование», также часто используется термин «дatalogическое проектирование», однако в рамках данного пособия будет использоваться именно первый вариант.

Разработка схемы или структуры БД включает в себя создание всех информационных объектов (таблиц) и связей между ними, определение их имен, типов данных, длин полей и т. д.

Корректность разработанной БД проверяется на основе анализа функциональных зависимостей между атрибутами таблиц и связана с теорией нормализации, рассматриваемой в п. 4.2.

Физическое проектирование связано с выбором эффективного размещения БД на внешних носителях с целью обеспечения максимально эффективного функционирования информационной системы [8].

3. Реализация информационной системы.

Данный этап завершает реализацию всех требований к информационной системе, описанных заказчиком, и заключается в решении ряда задач, а именно:

- разработка программного обеспечения;
- тестирование ИС;
- создание рабочей документации, инструкций пользователей по установке, настройке и эксплуатации.

Основной частью этапа является разработка программного обеспечения информационной системы, которое будет взаимодействовать с базой данных и обеспечивать пользовательский интерфейс. Кроме того, неотъемлемой частью завершения процесса разработки является обязательное тестирование всего программного комплекса на наличие неисправностей и ошибок. Конечно же, в любой системе, уже работающей на предприятии, неизбежно возникают сбои и ошибки при функционировании, т. к. полностью избежать наличия неисправностей в системе невозможно. Тем не менее, задачей разработчиков является минимизация ошибок в функционировании системы и повышение её надежности путём проведения целого ряда тестов для выявления максимально возможного числа ошибок.

По завершению этапа заказчик получает готовую к введению в эксплуатацию информационную систему, в обязательном порядке снабженную рабочей документацией по установке, настройке и эксплуатации программного комплекса.

4. Внедрение и эксплуатация ИС.

Завершают жизненный цикл этапы внедрения и эксплуатации информационной системы.

Этап внедрения или ввода системы в эксплуатацию включает в себя процессы установки и настройки программного обеспечения с использованием имеющегося в организации оборудования и системного программного обеспечения. Зачастую с этим шагом связаны процессы предварительной подготовки рабочих мест пользователей и технических средств. Ещё одним важным мероприятием является обучение персонала взаимодействию с пользовательским интерфейсом информационной системы. Завершается этап внедрения приемочными испытаниями системы, когда моделируется реальное взаимодействие пользователей с новой системой и та уже окончательно переходит в работоспособное состояние – этап эксплуатации.

Этап эксплуатации так же, как и остальные, состоит из набора процессов, связанных с ним. Основным из них является непосредственно сама эксплуатация системы, т. е. её использование для учёта и анализа имеющейся информации всеми пользователями.

В ходе эксплуатации системы неизбежно будут выявляться различные ошибки и сбои, которые могут быть связаны как с функционированием самой системы, так и с работой оборудования и имеющегося программного обеспечения, а также при их непосредственном взаимодействии. Вследствие этого на этапе эксплуатации важны процессы сопровождения системы или технической поддержки, которые позволят вовремя устранять возникшие проблемы и минимизировать время неработоспособного состояния системы.

Ещё одним важным процессом заключительного этапа является подготовка к совершенствованию и модернизации системы. Это связано с постепенным устареванием всего имеющегося оборудования в ходе его эксплуатации, а также появлением новых более совершенных технологий, программных и технических средств. В современном мире уже по прошествии 5–7 лет всё имеющееся программное и аппаратное обеспечение на предприятии может оказаться устаревшим, и вопрос модернизации может стать решающим для сохранения эффективности функционирования любой организации.

Этап эксплуатации системы завершается после полного прекращения использования системы и выполнения всех процессов, связанных с технической поддержкой и модернизацией.

4.2 Нормализация БД

При проектировании баз данных разработчик сталкивается с целым рядом ситуаций, в которых от выбора наиболее оптимального решения будет зависеть эффективность дальнейшего функционирования информационной системы. В некоторых ситуациях в зависимости от масштабов проекта могут быть выбраны различные варианты проектирования БД, и некоторые из перечисленных этапов жизненного цикла информационных систем не обязательно могут быть реализованы в полной мере.

Так, например, одним из классических методов проектирования баз данных является метод нормальных форм, или проектирование баз данных с использованием методов теории нормализации. Применение нормализации для поиска максимально эффективной структуры базы данных является одним из обязательных этапов при использовании любого подхода в проектировании

информационных систем. Однако использование только лишь методов нормализации при проектировании баз данных может значительно усложнить эту задачу, особенно для проектов с большим количеством отношений и связей. Таким образом, нормализацию БД лучше всего применять на этапе логического проектирования, когда уже имеющуюся структуру таблиц необходимо скорректировать для исключения ситуаций дублирования данных и возникновения различного вида аномалий.

Использование только метода нормализации для проектирования БД лучше всего подойдет для проектов с малым количеством объектов и связей. В этом случае вся информация объединяется в одно большое отношение, которое в дальнейшем путем декомпозиции разбивается на несколько взаимосвязанных.

4.2.1 Дублирование данных

В любой таблице БД может возникнуть ситуация дублирования данных. Как уже отмечалось, ситуации дублирования данных разработчику необходимо отслеживать и по возможности устранять. Однако не все ситуации дублирования отрицательно влияют на корректность БД. А потому важно отличать *простое* дублирование и *аномальное*, или избыточное, дублирование данных, т. е. дублирование, которое приводит к избыточности и аномалиям.

В качестве примеров рассмотрим две таблицы, содержащие сведения о заказах, сделанных клиентами у компании. Как видно из таблицы 4.1, один и тот же клиент сделал несколько заказов в разные дни, а потому значение его атрибута «ФИО» дублируется несколько раз. Или, например, в один и тот же день было сделано несколько заказов, а потому для всех записей об этом продублировано значение атрибута «Дата заказа». Данный пример иллюстрирует ситуацию простого дублирования, когда появление одинаковых данных неизбежно и не приводит к аномальным ситуациям.

Таблица 4.1 – Заказы (простое дублирование)

Номер (ПК)	ФИО Клиента	Дата заказа	Сумма заказа, руб.
568	Иванов С. П.	14.05.2017	22700
569	Петров И. С.	14.05.2017	42000
570	Иванов С. П.	16.05.2017	35200
571	Иванов С. П.	19.05.2017	25000

Во втором случае в аналогичной таблице добавим дополнительный атрибут, который будет хранить сведения о телефоне клиента. Данные из таблицы 4.2 наглядно характеризуют ситуацию аномального дублирования, которая и приводит к ситуациям неэффективной организации памяти, а также к аномалиям.

Таблица 4.2 – Заказы (аномальное дублирование)

Номер (ПК)	ФИО Клиента	Телефон клиента	Дата заказа	Сумма заказа, руб.
568	Иванов С. П.	445566	14.05.2017	22700
569	Петров И. С.	778899	14.05.2017	42000
570	Иванов С. П.	445566	16.05.2017	35200
571	Иванов С. П.	445566	19.05.2017	25000

Очевидно, что очередное появление номера телефона клиента, который регулярно делает заказы, ведет к тому, что это значение постоянно дублируется, хотя никакой необходимости в каждом повторении самого номера нет.

В качестве одного из возможных решений неопытный разработчик может предложить вариант однократного занесения номера телефона для каждого клиента, а в случае дальнейшего появления заказов от того же клиента значение атрибута «Телефон клиента» можно просто оставлять пустым, как это показано в таблице 4.3.

Таблица 4.3 – Заказы (аномальное дублирование с пустыми значениями)

Номер (ПК)	ФИО Клиента	Телефон клиента	Дата заказа	Сумма заказа, руб.
568	Иванов С. П.	445566	14.05.2017	22700
569	Петров И. С.	778899	14.05.2017	42000
570	Иванов С. П.	—	16.05.2017	35200
571	Иванов С. П.	—	19.05.2017	25000

Однако такой вариант решения проблемы не только не устраняет ситуации избыточности, но и приводит к другим сложностям. Во-первых, очевидно, что даже для пустых значений в любой записи в базе данных отводится определенное количество памяти, а потому незаполнение данных полей не приведет к

снижению занимаемого данными объема. Во-вторых, при большом объеме данных придется постоянно осуществлять поиск первой записи клиента с целью выяснения его номера телефона. В-третьих, в случае удаления первоначального заказа любого из клиентов информация о телефоне будет безвозвратно утрачена. Очевидно, что подобное решение приведёт только к ухудшению ситуации.

4.2.2 Аномалии



***Аномалия** – ситуация в таблицах БД, приводящая к противоречиям и усложняющая процессы обработки данных.*

Существует 3 вида аномалий:

- 1) аномалия удаления;
- 2) аномалия обновления;
- 3) аномалия добавления.



***Аномалия удаления** – ситуация, возникающая при удалении некоторых записей в таблицах БД, которая приводит к возможной потере данных, не предназначенных для удаления.*

Если из таблицы 4.2 удалить любой из заказов клиента Иванова, например, по причине неоплаты или отказа клиента, то это не приведет к потере информации о номере его телефона. Однако если удалить заказ Петрова по любой причине, то информация о номере его телефона исчезнет из базы данных, поскольку других заказов у Петрова нет.



***Аномалия обновления** – ситуация, возникающая при изменении некоторых записей в таблицах БД, которая приводит к возможному просмотру всех записей в таблицах БД, а также к изменению некоторых других записей.*

Так, изменение номера телефона у Иванова на новый приведёт к тому, что в таблице 4.2 потребуется просмотреть весь список заказов и в тех из них, в которых клиентом является Иванов, изменить номер телефона на новый.



***Аномалия добавления** – ситуация, возникающая при добавлении новых записей в таблицу БД, которая приводит к необходимости просмотра целого набора записей в таблице БД вследствие нехватки некоторой информации для внесения.*

В случае добавления нового заказа от клиента Петрова первоначально необходимо будет просмотреть таблицу и найти предыдущие заказы Петрова для того, чтобы внести уже имеющийся номер телефона в поле новой записи [3].

4.2.3 Теория нормализации

Теория нормализации основана на том, что определенный набор отношений (таблиц) обладает лучшими свойствами при добавлении, изменении и удалении данных, чем все другие наборы, с помощью которых можно представить те же самые данные.



***Нормализация** – процесс реорганизации данных путем устранения повторяющихся групп и других противоречий в организации данных с целью приведения набора таблиц к виду, позволяющему осуществлять корректные операции над данными.*

Воспользуемся данной теорией для решения проблемы аномального дублирования в примере из таблицы 4.2. Для этого проведем декомпозицию исходной таблицы на две новые, приведенные на рисунке 4.3.

Алгоритм декомпозиции, или нормализации, будет рассмотрен ниже, но стоит отметить, что размещение информации о клиенте в отдельной таблице в данном случае позволит избежать ситуаций избыточного дублирования данных и возникновения аномалий. Теперь удаление любого заказа не будет ни в одном из случаев приводить к потере информации о телефоне клиента, даже если удалить все записи из таблицы «Заказы». Проблема аномалии модификации не потребует просмотра всей таблицы и будет проведена лишь единожды в таблице «Клиенты». Изменится лишь порядок добавления информации при появлении новых клиентов: первоначально необходимо будет внести данные о новом клиенте в таблицу «Клиенты», а затем занести информацию о новом заказе этого клиента в таблицу «Заказы».



Рис. 4.3 – Таблицы «Заказы» и «Клиенты»

Как видно, данный набор таблиц хранит тот же самый объем данных, что и в таблице 4.2, однако он обладает лучшими свойствами. Подобная схема БД называется *корректной*.



***Корректной** называется схема БД, в которой отсутствуют нежелательные зависимости между атрибутами отношений.*

Теория нормализации основана на анализе функциональных зависимостей между атрибутами отношений. Понятие функциональной зависимости является фундаментальным в теории нормализации реляционных баз данных. Функциональные зависимости определяют устойчивые отношения между объектами и их свойствами в рассматриваемой предметной области.

Процесс проектирования с использованием декомпозиции представляет собой процесс последовательной нормализации схем отношений, при этом каждая последующая итерация соответствует нормальной форме более высокого уровня и обладает лучшими свойствами по сравнению с предыдущей.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- нормальная форма Бойса – Кодда (НФБК);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Основные свойства нормальных форм:

- каждая следующая нормальная форма «улучшает» свойства предыдущей;
- при переходе к каждой следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

В основе классического процесса нормализации лежит последовательность переходов от предыдущей нормальной формы к последующей, и так до последней. Однако на практике в большинстве случаев используются только первые три нормальные формы, т. е. нормализация проводится до достижения всеми таблицами схемы БД третьей нормальной формы.

Функциональные зависимости определяют не текущее состояние БД, а все возможные ее состояния, то есть они отражают те связи между атрибутами, которые присущи реальному объекту, который моделируется с помощью таблиц БД.

Поэтому определить функциональные зависимости по текущему состоянию БД можно только в том случае, если экземпляр БД содержит абсолютно полную информацию (то есть никаких добавлений и модификации БД не предполагается). В реальной жизни это требование невыполнимо, поэтому набор функциональных зависимостей задает разработчик или системный аналитик, исходя из глубокого системного анализа предметной области [8].



.....

*Понятие зависимости атрибутов или функциональной зависимости можно определить следующим образом: атрибут **В** функционально зависит от атрибута **А**, если каждому значению **А** соответствует в точности одно значение **В** (обозначается $A \rightarrow B$).*

.....

То есть во всех кортежах (записях) с одинаковым значением атрибута A атрибут B будет обязательно иметь также одно и то же значение. В качестве A и B могут выступать как простые атрибуты, так и составные, т. е. состоящие из нескольких атрибутов.

Так, в рассматриваемой выше таблице 4.2 очевидно имелась функциональная зависимость между атрибутами «ФИО клиента» и «Телефон клиента», которая и приводила к аномалиям и избыточности. В данном примере наличие такой зависимости привело к декомпозиции отношения и его преобразованию в два связанных отношения.

4.2.4 Нормальные формы

Как уже отмечалось, на практике процесс нормализации БД состоит в последовательном приведении исходных таблиц БД сначала к первой, затем ко второй и третьей нормальным формам.

Первая нормальная форма (1НФ)



.....

*Таблица находится в **первой нормальной форме** (в 1НФ), если все значения атрибутов являются простыми, а все записи – уникальными. Любая реляционная таблица по умолчанию находится в 1НФ.*

.....

В большинстве случаев процесс нормализации не потребует приведения таблицы к 1НФ, ведь все реляционные таблицы уже находятся в 1НФ. Тем не менее, возможны ситуации, когда некоторые абстрактные таблицы, не являющиеся реальными таблицами БД, необходимо нормализовать и, прежде всего, привести к 1НФ.

В качестве примера рассмотрим таблицу 4.4, хранящую информацию о заказах товаров. В данной таблице имеются данные о заказе (Номер и Дата), о клиенте, сделавшем заказ (ФИО и Телефон), а также о заказанных товарах и их количестве.

Таблица 4.4 – Заказы товаров (ненормализованная таблица)

Номер	ФИО Клиента	Телефон	Дата	Товары
568	Иванов С. П.	445566	14.05.2017	Стул – 10 шт. Стол – 3 шт.

Номер	ФИО Клиента	Телефон	Дата	Товары
569	Петров И. С.	778899	14.05.2017	Стул – 15 шт. Стол – 5 шт.
570	Иванов С. П.	445566	16.05.2017	Стул – 15 шт. Стол – 3 шт.
571	Иванов С. П.	445566	19.05.2017	Стул – 11 шт.

Несмотря на то что визуально данная таблица достаточно удобно представляет информацию, с точки зрения реляционной модели данных подобная организация недопустима. Прежде всего, нарушено правило атомарности значений, ведь поле «Товары» содержит не только набор значений товаров, но и их количество, указанное в этом же поле. Теоретически такую таблицу можно организовать в любой реляционной СУБД, просто создав текстовый атрибут «Товары» и занося в него подобные строки в ручном режиме. Однако в дальнейшем очень сложно будет производить простейшие операции запросов, например на вычисление общего количества проданных стульев за период и т. п. Очевидно, что с точки зрения реляционной модели данная таблица должна быть преобразована за счет разделения атрибута «Товары», а также путем добавления новых строк для каждого товара в одном заказе. Полученная таблица приведена ниже (табл. 4.5).

Таблица 4.5 – Заказы товаров в 1НФ

Номер (ПК)	ФИО Клиента	Телефон	Дата заказа	Товар (ПК)	Кол-во, шт.
568	Иванов С. П.	445566	14.05.2017	Стул	10
568	Иванов С. П.	445566	14.05.2017	Стол	3
569	Петров И. С.	778899	14.05.2017	Стул	15
569	Петров И. С.	778899	14.05.2017	Стол	5
570	Иванов С. П.	445566	16.05.2017	Стул	15
570	Иванов С. П.	445566	16.05.2017	Стол	3
571	Иванов С. П.	445566	19.05.2017	Стул	11

Данная таблица очевидно аномально избыточна, ведь значения полей «ФИО Клиента», «Телефон» и «Дата заказа» будут дублироваться для всех строк одного заказа, однако соответствует требованиям реляционной модели, а

потому находится в 1НФ. Обратим внимание, что в новой таблице первичный ключ является составным и включает 2 атрибута: «Номер» и «Товар». Очевидно, что простой ПК «Номер», ранее обозначавший номер каждого заказа, теперь не может быть использован, т. к. в новой структуре для него нарушается правило уникальности. Один и тот же «старый» ПК теперь был бы у всех записей, в которых товаров перечислено было бы больше одного. Эта проблема была решена путём добавления в состав ключа атрибута «Товар».

Важно отметить, что для каждой таблицы, подлежащей нормализации, на данном этапе обязательно необходимо определить набор атрибутов, которые будут составлять первичный ключ. Все атрибуты, не вошедшие в состав ПК, будут называться неключевыми.

Вторая нормальная форма (2НФ)



.....

*Таблица находится во **второй нормальной форме (2НФ)**, если она находится в 1НФ и её неключевые атрибуты полностью зависят от всего первичного ключа. Другими словами, в таблице не должно быть неключевых атрибутов, которые зависят от части первичного ключа.*

.....

Рассмотренная выше таблица 4.5 не находится в 2НФ, поскольку в ней имеются атрибуты, которые зависят от части первичного ключа. Поля «ФИО Клиента», «Телефон» и «Дата заказа» зависят от поля «Номер», которое является только частью ПК, и не связаны с тем, сколько и каких товаров заказал клиент. Поле «Кол-во» зависит от всех частей ПК одновременно, поскольку количество даже одного и того же товара в разных заказах может оказаться разным.

Для перевода таблицы из 1НФ в 2НФ необходимо воспользоваться следующим алгоритмом:

1. Определить все частичные зависимости, т. е. все части первичного ключа и все неключевые атрибуты, которые от них зависят.
2. Для каждой части первичного ключа и набора неключевых атрибутов, которые от них зависят, необходимо создать новую таблицу и скопировать эти данные в неё. В новой таблице бывшая часть первичного ключа станет первичным ключом.

3. Из исходной таблицы необходимо удалить все скопированные неключевые атрибуты, при этом не удаляя части первичного ключа, которые станут ещё и внешними ключами, связанными с новыми таблицами.

В нашем примере мы уже выделили зависимость некоторых неключевых атрибутов от поля «Номер» и согласно алгоритму должны создать новую таблицу для всей этой группы атрибутов вместе с частью первичного ключа.

В исходной таблице данную группу неключевых атрибутов необходимо удалить, оставив только поле «Номер». Данный атрибут теперь также будет являться и внешним ключом.

В результате приведения к 2НФ будет получено две таблицы, связанные по атрибуту «Номер» (рис. 4.4).

«Заказы»

Номер (ПК)	ФИО Клиента	Телефон	Дата
568	Иванов С. П.	445566	14.05.2017
569	Петров И. С.	778899	14.05.2017
570	Иванов С. П.	445566	16.05.2017
571	Иванов С. П.	445566	19.05.2017

«Заказы товаров»

Номер (ПК, ВК)	Товар (ПК)	Кол-во, шт.
568	Стул	10
568	Стол	3
569	Стул	15
569	Стол	5
570	Стул	15
570	Стол	3
571	Стул	11

Рис. 4.4 – Таблицы «Заказы» и «Заказы товаров» в 2НФ



Любая реляционная таблица, у которой первичный ключ состоит из одного атрибута или является простым, по умолчанию

находится в 2НФ, поскольку первичный ключ такой таблицы невозможно разделить на части.

.....

Третья нормальная форма (3НФ)

.....



*Таблица находится в **третьей нормальной форме (3НФ)**, если она находится в 2НФ и её неключевые атрибуты зависят только от первичного ключа. Другими словами, в таблице не должно быть неключевых атрибутов, которые зависят от других неключевых атрибутов.*

.....

После приведения к 2НФ в таблице «Заказы» на рисунке 4.4 по-прежнему видно anomalous дублирование. Следовательно, данная таблица находится в 2НФ, но не в 3НФ. Действительно, здесь имеется функциональная зависимость между атрибутами «ФИО Клиента» и «Телефон», которая рассматривалась в пп. 4.2.1.

Для того чтобы перевести таблицу из 2НФ в 3НФ, необходимо воспользоваться следующим алгоритмом:

1. Определить зависимости группы одних неключевых атрибутов от других неключевых атрибутов.
2. Для каждой такой неключевой атрибута и набора неключевых атрибутов, которые от него зависят, необходимо создать новую таблицу и скопировать эти данные в неё. В новой таблице бывший неключевой атрибут, от которого зависят остальные, станет первичным ключом.
3. Из исходной таблицы необходимо удалить все скопированные неключевые атрибуты, при этом не удаляя того, от которого все зависят, ведь он станет ещё и внешним ключом, связанным с новой таблицей.

В нашем примере мы уже выделили зависимость неключевого атрибута «Телефон клиента» от поля «ФИО Клиента» и согласно алгоритму должны создать новую таблицу для всей этой группы атрибутов.

В исходной таблице данную группу неключевых атрибутов необходимо удалить, оставив только поле «ФИО Клиента». Данный атрибут теперь также будет являться и внешним ключом.

В результате приведения к 3НФ будут получены таблицы, изображенные на рисунке 4.5.

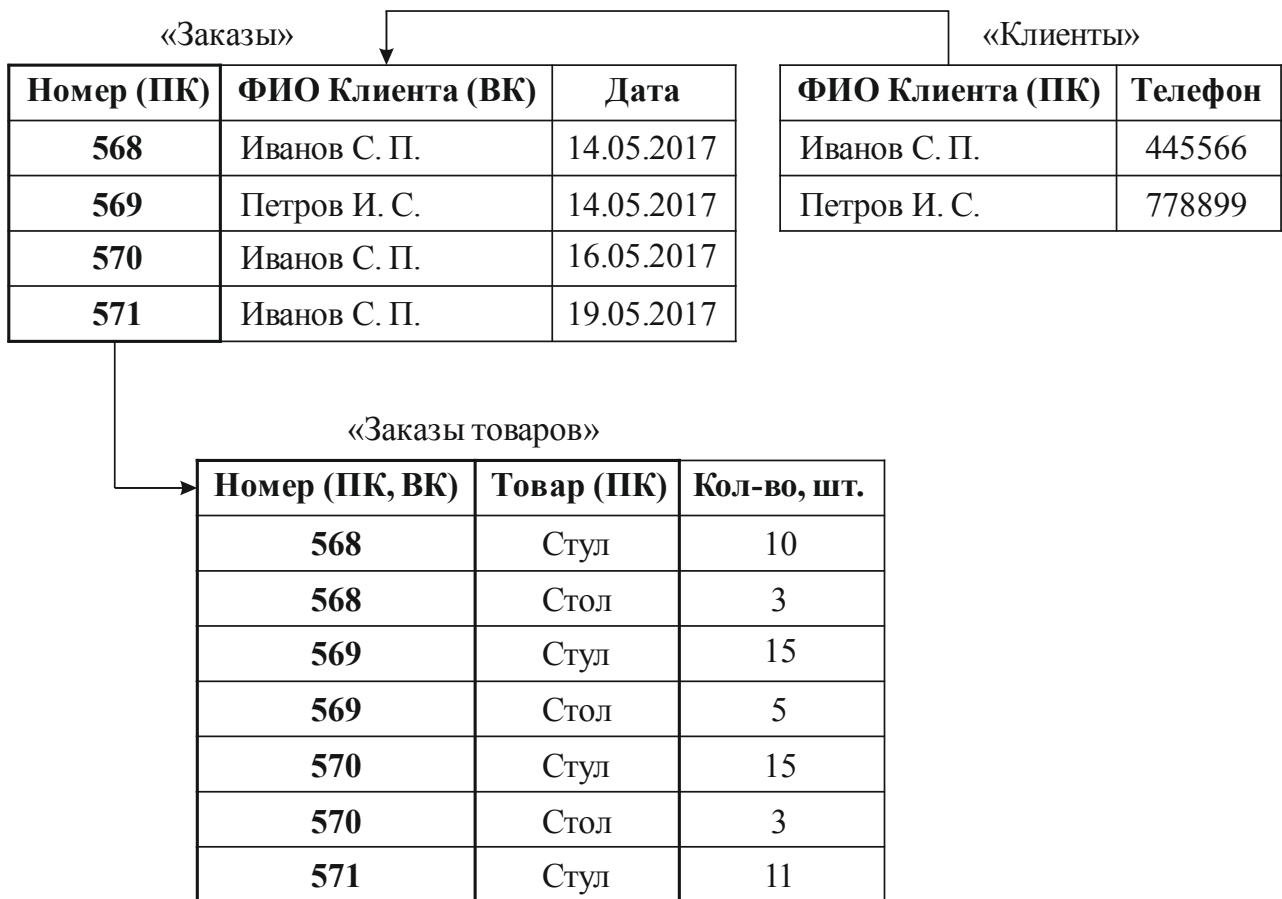


Рис. 4.5 – Таблицы «Заказы», «Заказы товаров» и «Клиенты» в 3НФ

Полученная схема БД считается нормализованной и готовой к следующим этапам проектирования. Нормализация позволила устранить избыточность данных, что в свою очередь приведёт к снижению объема хранимых данных и устранит различные аномалии.

Пример нормализации данных

В качестве примера нормализации данных рассмотрим информацию об успеваемости студентов, содержащуюся в таблице 4.6. Данную таблицу необходимо привести к третьей нормальной форме. В таблице «Успеваемость» хранится информация о студентах, сдавших экзамен, предмете, номере группы студента, ФИО преподавателя, принимавшего экзамен, а также дата сдачи экзамена и итоговая оценка по предмету.

На первом этапе необходимо определить соответствие таблицы первой нормальной форме. В данном случае таблица уже находится в 1НФ, поскольку все значения полей простые или атомарные, а все записи уникальные. Прежде чем проводить дальнейшую нормализацию, для любой таблицы необходимо выбрать атрибут или набор атрибутов, которые будут выступать в качестве

первичного ключа. В данной таблице нельзя выбрать простой первичный ключ, так как ни один атрибут не уникален. Проводя анализ таблицы, нельзя забывать о том факте, что имеющийся набор данных не окончательный и в дальнейшем в таблицу может быть добавлено большое число новых записей, а потом выбор первичного ключа нужно осуществлять, предполагая различные варианты добавления новых записей.

Таблица 4.6 – Успеваемость

Студент	Предмет	Группа	Преподаватель	Дата сдачи экзамена	Оценка
Иванов С. П.	Базы данных	598	Белов А. В.	14.06.2016	4
Петров И. С.	Базы данных	598	Белов А. В.	14.06.2016	5
Иванов С. П.	Информатика	598	Алексеев А. С.	18.06.2016	4
Петров И. С.	Информатика	598	Алексеев А. С.	18.06.2016	4

В данном случае выбор пары атрибутов («Студент», «Предмет») в качестве ПК будет наиболее оптимальным, ведь каждым студентом по каждому предмету будет получена лишь одна оценка (см. табл. 4.7).

Таблица 4.7 – Успеваемость в 1НФ

Студент (ПК)	Предмет (ПК)	Группа	Преподаватель	Дата сдачи экзамена	Оценка
Иванов С. П.	Базы данных	598	Белов А. В.	14.06.2016	4
Петров И. С.	Базы данных	598	Белов А. В.	14.06.2016	5
Иванов С. П.	Информатика	598	Алексеев А. С.	18.06.2016	4
Петров И. С.	Информатика	598	Алексеев А. С.	18.06.2016	4

Следующим этапом нормализации будет проверка таблицы на соответствие второй нормальной форме. Здесь необходимо установить возможные функциональные зависимости между неключевыми атрибутами и частями первичного ключа. Проведя анализ, нетрудно заметить, что таблица не находится в 2НФ, поскольку имеются неключевые атрибуты, зависящие от части ПК. Так, атрибут «Группа» зависит от ключевого атрибута «Студент» и не зависит от предмета, который сдавал студент. Другой атрибут «Преподаватель», наоборот,

зависит от «Предмета» и не связан с тем, кто и когда сдавал экзамен. Атрибуты «Дата сдачи экзамена» и «Оценка» связаны со всем ключом полностью.

Таким образом, для перехода к 2НФ необходимо для каждой группы, зависящей от части ключа, создать новую таблицу и удалить из исходной таблицы неключевые атрибуты. В результате будет получена следующая схема БД, изображенная на рисунке 4.6.



Рис. 4.6 – Схема БД «Успеваемость» в 2НФ

Далее необходимо проверить все полученные таблицы на соответствие третьей нормальной форме, т. е. проанализировать возможные функциональные зависимости между неключевыми атрибутами. В данном примере только в одной таблице имеется два неключевых атрибута, между которыми возможна зависимость, однако «Оценка» и «Дата сдачи экзамена» между собой никак не связаны, а потому во всех таблицах нет несоответствия 3НФ.

Таким образом, можно сделать вывод, что таблицы на рисунке 4.6 находятся одновременно и в 2НФ, и в 3НФ и дальнейшая декомпозиция не требуется. Нормализацию в данном примере можно считать завершенной.

4.3 Инфологическое проектирование БД

Несмотря на то что исторически нормализация считается одним из классических методов проектирования БД, её методы лучше использовать на этапе логического проектирования с целью оценки корректности полученной ранее схемы баз данных. В настоящее время для построения инфологической модели базы данных наиболее часто используется неформальная модель предметной

области, получившая название *модель «сущность – связь»*, или *ER-модель*. Данная модель позволяет описывать объекты предметной области и их взаимоотношения на основе семантического (смыслового) описания предметной области и представления информации в удобном графическом варианте для дальнейшей реализации в виде базы данных.

4.3.1 Модель «сущность – связь»

Модель «сущность – связь», часто также называемая ER-модель (от. англ. *Entity-Relationship*), была предложена П. Ченом ещё в 1976 г. и представляет собой одну из самых известных концептуальных моделей для описания предметной области. ER-модель основана на использовании трёх базовых понятий: *сущность*, *атрибут* и *связь*.

Сущность – множество реально существующих или абстрактных объектов, процессов или явлений, обладающих одинаковым набором свойств, называемых атрибутами.

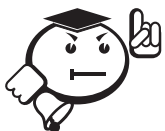
Ещё раз уточним, что важно понимать и отличать термины *сущность* и *экземпляр сущности*. *Сущность* определяет набор однородных объектов, а *экземпляр сущности* – это конкретный объект из данного набора.

Атрибут – это поименованное свойство (характеристика) сущности. Основное назначение атрибута – описание свойства сущности, а также идентификация каждого экземпляра сущности.

Связь – ассоциация между сущностями, которая показывает, каким образом сущности взаимодействуют или соотносятся между собой. Чаще всего связь возникает между двумя сущностями и показывает, каким образом связаны экземпляры сущностей между собой.

ER-диаграммы

ER-модель чаще всего представляется в графическом виде. Для этого автором ER-модели была придумана нотация, называемая ER-диаграмма.



В настоящее время понятия «ER-модель» и «ER-диаграмма» стали практически синонимами, хотя для графического представления ER-моделей могут быть использованы и другие нотации, например Crow's Foot, IE – Information Engineering, IDEF1X и др. Тем не менее, на практике графическое представление ER-модели с

использованием любой из нотаций принято называть ER-диаграммой.

.....

В данном пособии для обозначения сущностей и атрибутов ER-модели будут использоваться обозначения, изображенные на рисунке 4.7.

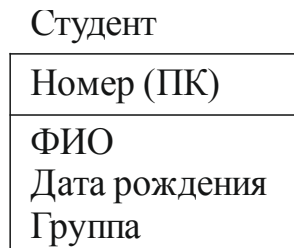


Рис. 4.7 – Обозначение сущности на ER-диаграмме

Сущность обозначается прямоугольником, при этом название сущности указывается над ним. Список атрибутов размещается внутри прямоугольника и разделён одной горизонтальной чертой. Сверху от неё расположены ключевые атрибуты, т. е. атрибут или набор атрибутов, составляющих первичный ключ. Все остальные неключевые атрибуты расположены под чертой. Данный набор атрибутов представляет собой фактически схему будущего отношения реляционной модели. В данном примере рассматривается сущность «Студент» с атрибутами «Номер» (первичный ключ), «ФИО», «Дата рождения» и «Группа».

В большинстве случаев сущности связаны друг с другом. Связь может существовать между двумя разными сущностями, также сущность может быть связана сама с собой в случае рекурсивной связи. В данном пособии для обозначения связей между сущностями будут использоваться обозначения, изображенные на рисунках 4.8 и 4.9.

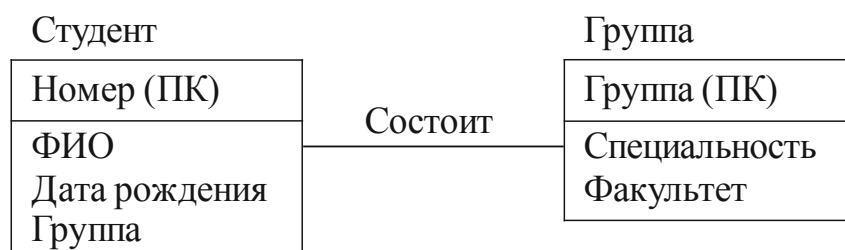


Рис. 4.8 – Обозначение связей между сущностями

На рисунке 4.8 приведён пример связи, уже рассмотренной в данном пособии, между сущностями «Студент» и «Группа». На первых этапах проектирования для однозначной интерпретации информации о предметной области

связи обозначают «глагольными фразами» для понимания особенностей отношения между сущностями. Во многих случаях именование фразы осуществляется в двух направлениях, хотя это требование и не является обязательным. Более опытные проектировщики могут использовать только один наиболее удобный и короткий глагол или не использовать такие фразы совсем, поскольку в большинстве случаев в этом нет необходимости в силу очевидности связей.

В данном примере связь между двумя сущностями можно определить следующим образом: «студент состоит в группе» или «группа состоит из студентов». Также в этом примере можно использовать фразы «обучается» или «учится», ведь важно понимать, что основная суть подобных обозначений состоит в удобном и наглядном отображении информации о предметной области в виде диаграмм для заказчиков информационных систем, не разбирающихся в тонкостях инфологического проектирования.

Поскольку все связи можно разделить на несколько категорий, дополнительно существует определённый набор обозначений для каждой из связей, однозначно интерпретирующий тип связи на диаграмме. Например, связь между сущностями «Студент» и «Группа» можно представить более подробно на рисунке 4.9.

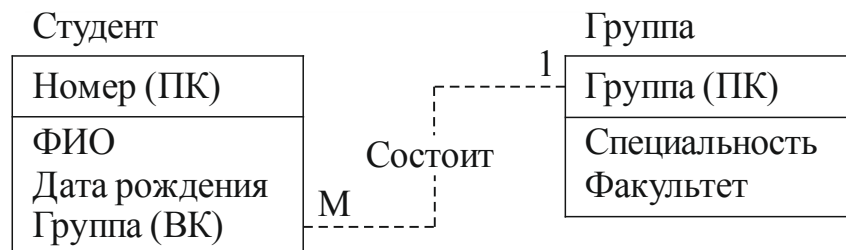


Рис. 4.9 – Обозначение вида связи между сущностями

Подобные обозначения связей можно встретить на ER-диаграммах уже ближе к концу этапа инфологического проектирования, когда проектировщик готовится к реализации полученной ER-модели к виду, в дальнейшем преобразованному в готовую БД.

На рисунке 4.9 приведен пример определенной неидентифицирующей связи вида 1:M между сущностями. Ещё одним небольшим отличием является размещение линии связи от одной сущности к другой, именно между связываемыми атрибутами «Группа». Это позволяет более наглядно определять внешние ключи в сущностях. Во многих ситуациях одна сущность может участвовать одновременно в двух и более связях с другими сущностями и количество

внешних ключей может быть значительным, а потому для проектировщика важно понимать какие атрибуты участвуют в одних связях, а какие – в других.

Более подробное описание видов связей и их обозначений на ER-диаграммах приведено далее.

4.3.2 Виды связей между сущностями

Все связи между сущностями в ER-модели можно классифицировать по нескольким критериям:

- 1) количество участвующих в связи сущностей;
- 2) мощность;
- 3) обязательность;
- 4) зависимость.

По количеству участвующих в связи сущностей все связи можно разделить на *бинарные*, т. е. связи между двумя различными сущностями, и *рекурсивные*, когда сущность связана сама с собой. Все рассмотренные выше связи между сущностями являются бинарными. Тем не менее, в БД иногда встречаются и рекурсивные связи, когда один из неключевых атрибутов сущности связан с первичным ключом этой же сущности. Пример рекурсивной связи приведен на рисунке 4.10.



Рис. 4.10 – Пример рекурсивной связи

Для хранения сведений обо всех сотрудниках организации создана сущность «Сотрудник». При этом у каждого сотрудника имеется начальник, которым очевидно является другой сотрудник организации. В подобных ситуациях и возникает рекурсивная связь, приведенная на рисунке.

Также все связи можно разделить по критерию *мощности*.



Мощность связи – количество экземпляров дочерней сущности, которое может существовать для каждого экземпляра родительской сущности.

Ранее нами уже были рассмотрены такие виды связей как «один-к-одному», «один-ко-многим» и «многие-ко-многим», которые являются наиболее простыми и понятными для классификации связей. Однако в различных нотациях классификация связей по мощности производится на более детализированном уровне. Например, в IDEF1X связи по мощности делятся на следующие:

- 1) каждый экземпляр родительской сущности может иметь 0, 1 или более связанных с ним экземпляров дочерней сущности;
- 2) каждый экземпляр родительской сущности должен иметь не менее одного связанного с ним экземпляра дочерней сущности;
- 3) каждый экземпляр родительской сущности должен иметь не более одного связанного с ним экземпляра дочерней сущности;
- 4) каждый экземпляр родительской сущности связан с фиксированным числом экземпляров дочерней сущности [7].

В данном пособии подобная классификация приведена в качестве примера и использоваться на практике не будет. Для любой связи будет просто указываться 1 или М для каждой из участвующих в связи сущностей, как это показано на рисунке 4.9.

По признаку *обязательности* связь может быть *обязательной*, если в данной связи должен участвовать каждый экземпляр сущности, или *необязательной*, если не каждый экземпляр сущности должен участвовать в связи. Кроме того, связь может быть обязательной с одной стороны и необязательной – с другой.

По признаку *зависимости* сущностей связи делятся на идентифицирующие и неидентифицирующие.



Идентифицирующая связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями.

Экземпляр зависимой (дочерней) сущности определяется только через отношение к родительской сущности. При установлении идентифицирующей

связи первичный ключ родительской сущности мигрирует в состав первичного ключа дочерней сущности. В этом и есть смысл термина «зависимая сущность» – поскольку внешний ключ становится частью первичного, то сущность просто не может существовать без указания значения внешнего ключа.

Классическим примером идентифицирующей связи является связь между сущностями «Заказ» и «Заказ Товаров», изображенная на рисунке 4.11, а также на рисунке 4.4, где связь и сущности уже реализованы в виде связанных реляционных таблиц. В данном примере связывается информация о заказе клиента с информацией о том, какие товары и в каком количестве входят в этот заказ.

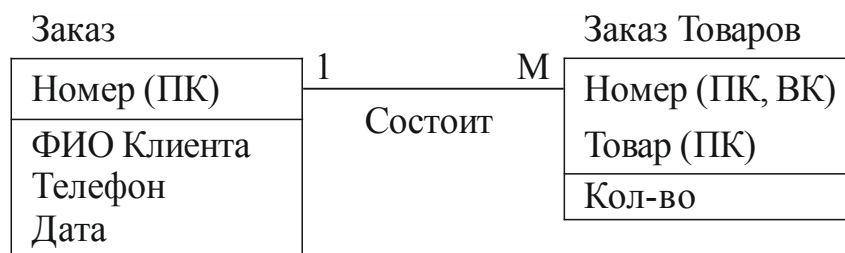


Рис. 4.11 – Пример идентифицирующей связи

Как видно из рисунка, идентифицирующая связь изображается сплошной линией. Сущность «Заказ Товаров» в данном примере является «зависимой», и во многих нотациях изображается прямоугольником со скругленными углами. Очевидно, что экземпляры зависимой сущности определяются только через отношение к родительской сущности, т. е. не могут существовать сами по себе, если не существует связанных с ними экземпляров-родителей. В данном случае товары в заказ не могут быть добавлены, пока не существует самого заказа, т. е. первичный ключ дочерней сущности не может быть определён, поскольку в его состав входит внешний ключ.



.....

*При установлении **неидентифицирующей** связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых атрибутов дочерней сущности.*

.....

Неидентифицирующая связь изображается пунктирной линией и встречается намного чаще, чем идентифицирующая. В неидентифицирующей связи экземпляры дочерней сущности могут существовать независимо от родительской,

поскольку мигрировавшие атрибуты родительской сущности не входят в состав первичного ключа.

На рисунке 4.12 изображен пример неидентифицирующей связи между сущностями «Заказ» и «Клиент». Сущность «Заказ» является независимой, поскольку атрибут «КодКлиента» является простым внешним ключом, который мигрировал из родительской сущности.

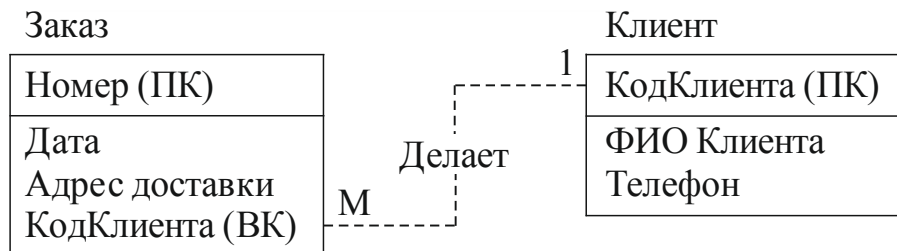


Рис. 4.12 – Пример неидентифицирующей связи

Кроме приведенных в классификации видов связи выделим ещё несколько важных терминов, которые в большей степени относятся к нотации IDEF1X, но могут встретиться в литературе в ситуациях проектирования предметной области на основе ER-моделей.



Определенной связью называется отношение между сущностями, в котором каждый экземпляр родительской сущности связан с 0, 1 или более экземплярами дочерней сущности и каждый экземпляр дочерней сущности связан с 0 или 1 экземпляром родительской сущности. Другими словами, определенной называется связь типа 1:1 или 1:М.

Определенные связи между сущностями реализуются посредством миграции первичного ключа родительской сущности в дочернюю, т. е. в дочерней сущности появляется атрибут – внешний ключ. Пример определенной связи приведен на рисунке 4.12.



Неопределенной связью называется отношение между двумя сущностями, при котором каждый экземпляр первой сущности связан с 0, 1 или более экземплярами второй сущности и каждый экземпляр второй сущности связан с 0, 1 или более экземплярами

первой сущности. Другими словами, определенной называется связь типа $M:M$.

.....

Неопределенные связи между сущностями реализуются на практике с помощью добавления новой сущности и связывания двух имеющихся с новой с помощью определенных связей (рис. 4.13). Механизм преобразования неопределенных связей в определенные заключается в следующем:

- 1) каждая неопределенная связь преобразуется в одну новую сущность, которая является дочерней по отношению к связываемым сущностям;
- 2) новой сущности присваивается уникальное имя;
- 3) между новой сущностью и каждой из связываемых сущностей устанавливается идентифицирующая определенная связь; каждой из этих связей назначается имя;
- 4) первичные ключи обеих родительских сущностей мигрируют в область первичного ключа новой сущности.

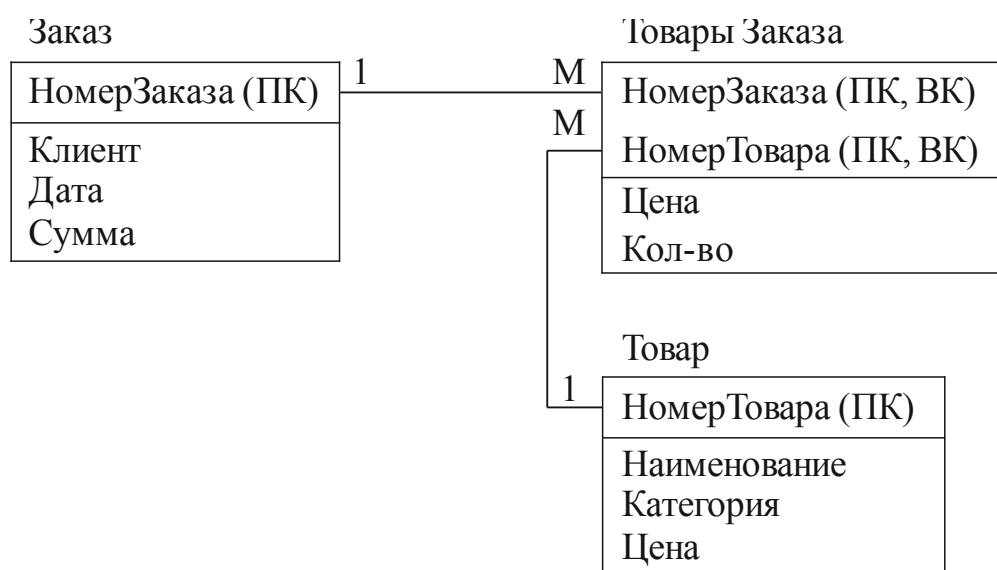


Рис. 4.13 – Реализация неопределенной связи между сущностями «Заказ» и «Товар»

Связь типа «Категория» встречается в реальном мире, когда некоторые сущности являются определенными категориями других сущностей. Например, «Преподаватель» – это определенная категория сущности «Сотрудник». Такие сущности, которые называются сущностями подтипа, хорошо использовать для хранения информации, относящейся только к какому-то конкретному подтипу. Сущность, на основе которой строятся подтипы, называется супертипом и содержит в себе атрибуты общие для всех подтипов.

Рассмотрим в качестве примера предметную область библиотеки вуза, в которой имеется сущность «Читатель» с атрибутами (Код, ФИО, Дата рождения, Тип). Очевидно, что в качестве читателей библиотеки могут выступать как студенты, так и преподаватели вуза, имеющие разные наборы атрибутов. Таким образом, сущности «Преподаватель» и «Студент» можно считать подтипами супертипа «Читатель». Важно отметить, что сущности типа «Категория» возникают только в тех ситуациях, когда подтипы имеют отличающийся друг от друга набор атрибутов. Например, на рисунке 4.14 у преподавателей имеются такие атрибуты, как «Должность», «Кафедра», «Ученая степень», а у студентов – «Факультет», «Номер группы» и т. д.

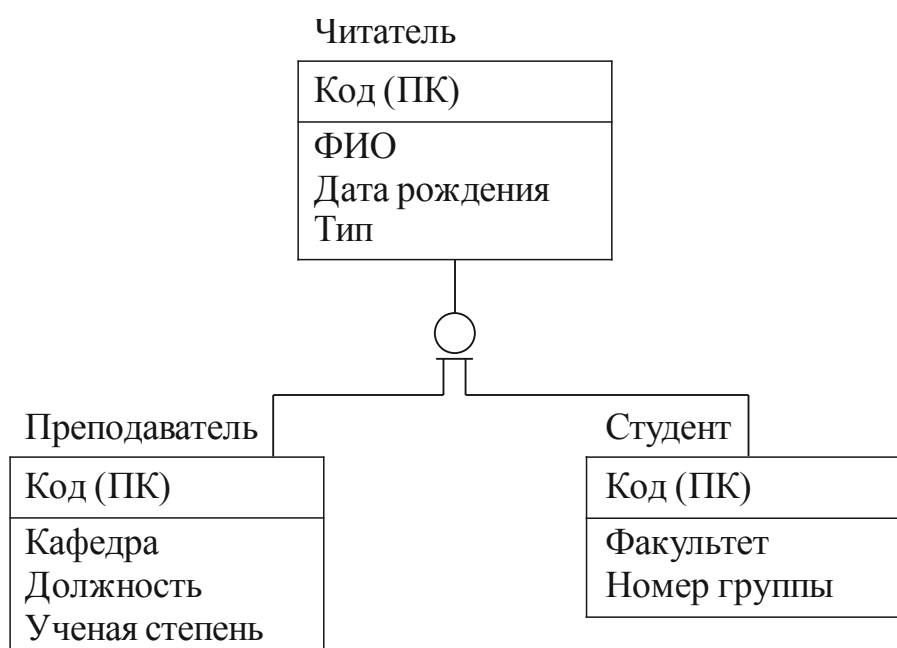


Рис. 4.14 – Пример связи типа «Категория»

Для реализации связи типа «Категория» первичный ключ сущности-супертипа мигрирует в область первичных ключей всех сущностей подтипа, участвующих в этой связи. При этом первичный ключ сущности подтипа не имеет других атрибутов, кроме мигрировавших. Все связи между супертипом и подтипами относятся к виду 1:1.

В некоторых случаях все сущности связи типа «Категория» объединяют в единую, содержащую полный набор атрибутов всех сущностей-подтипов. Это может делаться с целью упрощения дальнейшего проектирования и реализации запросов к будущим таблицам. Такая ситуация поможет уменьшить общее число записей в БД, тем не менее важным недостатком может стать возрастание занимаемого данными объема. Следствием подобной реализации будет наличие

некоторого числа пустых ячеек, для которых в любом случае будет выделена память. Примером подобной реализации служит таблица 4.8. полученная путем объединения сущностей «Читатель», «Преподаватель» и «Студент».

Таблица 4.8 – Реализация связи типа «Категория»

Код	ФИО	Дата рождения	Тип	Ка-федра	Долж-ность	Ученая сте-пень	Фа-культет	Номер груп-пы
1	Иванов Сергей Петрович	14.05.1988	ст.				ЭТ	116
2	Петров Иван Сергеевич	27.01.1989	ст.				ВС	598
3	Сергеев Илья Борисович	15.11.1963	пр.	ЭМИС	ст. преп.			
4	Семенов Иван Дмитриевич	01.06.1974	пр.	АОИ	доцент	к.т.н.		
5	Алексеев Семен Оле-гович	05.10.1993	ст.				Экон.	446
6	Белов Петр Иванович	01.11.1988	ст.				ЭТ	116

4.3.3 Методология проектирования на основе ER-моделей

Основная суть проектирования состоит в последовательном переходе в представлении информации о предметной области от описательной формы к форме, близкой к модели базы данных. В качестве основной методологии проектирования, иллюстрирующей подход, основанный на использовании ER-диаграмм, далее используется стандарт моделирования данных IDEF1X, который представляет информацию в виде трех уровней логических моделей:

- 1) модель зависимости сущностей;
- 2) модель, основанная на ключах;
- 3) полноатрибутная модель.

Модель уровня сущностей является моделью нижнего уровня и применяется для работы проектировщика информационной системы с экспертами про-

ектируемой системы. Модель включает сущности и связи между ними, которые отражают основные объекты и их взаимодействие в предметной области, и допускает присутствие всех типов связей (определенных, неопределенных, типа «Категория»). Графическое представление этой модели называется ER-диаграммой.

Модель, основанная на ключах, является дальнейшим развитием ER-модели и содержит более подробное представление данных. Она содержит описание всех сущностей, связей между ними, первичных и внешних ключей. Эта модель не допускает наличия неопределенных связей и требует их предварительного преобразования в определенные связи. Модель является переходным звеном от модели уровня сущностей к полному описанию предметной области. Графическое представление этой модели называется KB-диаграммой (Key Based Diagram).

Полноатрибутная, или полностью определенная, модель является дальнейшим развитием модели уровня ключей и представляет собой законченное описание предметной области. Модель содержит описание всех сущностей, связей и атрибутов предметной области. Построением этой модели завершается процесс инфологического проектирования, а модель в дальнейшем может быть использована при построении готовой базы данных. Графическое представление этой модели называется FA-диаграммой (Full Attributed Diagram).

В соответствии с методологией стандарта IDEF1X построение итоговой ER-диаграммы любой предметной области состоит в последовательном построении всех трёх моделей поэтапно. Рассмотрим процесс построения модели предметной области на примере построения базы данных «Торговая компания».

Этап 1. Построение модели зависимостей сущностей

Первым шагом проектирования на данном этапе является составление предварительного списка сущностей. Сначала на основе анализа предметной области определяются так называемые кандидаты в сущности. Для этого выявляют экземпляры объектов предметной области с одинаковыми характеристиками и объединяют их в одну сущность-кандидат.

Для того чтобы выбрать среди объектов, являющихся кандидатами, собственно сущности, разработчик модели должен относительно каждого кандидата ответить положительно на следующие вопросы:

- Можно ли описать этот объект, т. е. можно ли получить о нем информацию?

- Имеет ли этот объект характеризующие его свойства?
- Можно ли получить информацию об этих свойствах?
- Можно ли выделить несколько образцов этого объекта, т. е. набор экземпляров этого объекта с одинаковыми свойствами?
- Можно ли отличить один образец этого объекта от другого, т. е. имеется ли у объекта свойство (группа свойств), определяющее уникальность каждого образца этого объекта?

Если некоторые из этих вопросов будут иметь отрицательный ответ, то, возможно, данного кандидата нужно рассматривать лишь в качестве атрибута другой сущности.

Результат работы на данном этапе проектирования можно оформить в виде таблицы (табл. 4.9). Здесь требуется отметить, что не все сущности из списка, составленного на первом этапе проектирования, могут остаться в пуле сущностей к концу этапа № 3. Кроме того, на следующих фазах проектирования в пул могут быть добавлены новые сущности.

Таблица 4.9 – Пул сущностей первого этапа

№	Имя сущности	Описание сущности
1.	Покупатель	Лицо, купившее у компании товары
2.	Товар	Предметы, продаваемые компанией
3.	Накладная	Документ, оформленный на единовременную закупку некоторого количества разных товаров одним клиентом

В нашем примере анализ предметной области показал, что Торговая компания занимается оптовой продажей товаров покупателям. На каждого покупателя в обязательном порядке оформляется документ – накладная на список приобретенных покупателем товаров.

После составления списка сущностей необходимо выявить всевозможные бинарные связи между ними. Каждой выявленной связи назначаются имя, которое является глаголом или глагольной фразой, а также тип связи – 1:1, 1:M, M:N (определенная/неопределенная). На данном этапе не всегда можно определить, является связь идентифицирующей или неидентифицирующей, а потому такое решение откладывается на последующие этапы проектирования.

Кроме того, для каждой из сущностей – участниц связи необходимо определить, является ли сущность родительской, или дочерней в указанной свя-

зи. Выбор родительских и дочерних сущностей влияет на определение глагольных имен связей. В случаях определенных связей имя связи является глаголом или глагольной фразой, связывающей имена сущностей – участниц связей в направлении от родительской к дочерней. В случае неопределенной связи имя связи задается либо двумя глаголами, которые связывают имена сущностей в обоих направлениях связи, либо в одном из выбранных проектировщиком направлений.

Последним шагом этапа является создание ER-диаграммы предметной области, полученной на основе имеющегося пула сущностей и выявленных между сущностями связей. Диаграмма Торговой компании приведена на рисунке 4.15.

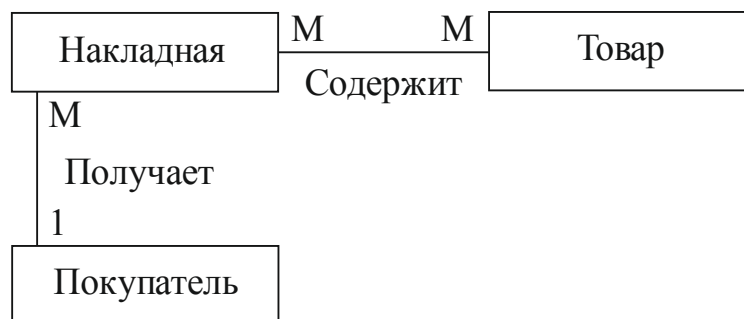


Рис. 4.15 – ER-диаграмма Торговой компании

В нашем примере имеются связи между сущностями «Покупатель» и «Накладная», а также между сущностями «Накладная» и «Товар». Каждый покупатель может получить много накладных, делая закупки в разные дни, но каждая накладная оформляется только на одного покупателя. В данном случае связь между сущностями определенная, или «один-ко-многим».

Во втором случае связь является неопределенной, или вида «многие-ко-многим», ведь каждая накладная может содержать много товаров, и каждый товар может быть указан в разных накладных.

Этап 2. Построение модели уровня ключей

Модель уровня ключей, или модель, основанная на ключах, является дальнейшим развитием модели зависимостей сущностей. На втором этапе основной задачей проектировщика является определение всех первичных и внешних ключевых полей каждой сущности. Для достижения такой цели на данном этапе необходимо осуществить следующие действия:

- для каждой сущности выбрать первичный ключ;

- реализовать все связи между сущностями;
- разрешить все неопределенные связи с помощью дополнительных сущностей.

Первым шагом второго этапа является выбор атрибутов в качестве первичных ключей для каждой сущности. Чаще всего в качестве первичного ключа выбирается некоторый уникальный номер или идентификатор, который в дальнейшем будет отличать каждый экземпляр сущности от другого. Для покупателя в качестве такового можно выбрать, например, номер ИНН, а для товара – номер на штрих-коде. Кроме того, можно задать для большинства сущностей собственную нумерацию на основе так называемого поля «Счетчик», который для каждой новой записи просто выбирает очередной порядковый номер, ведь подобный тип данных реализован в большинстве СУБД.

Далее для всех видов связей, кроме неопределенных, необходимо реализовать миграцию первичных ключей родительских сущностей в дочерние. Для каждой определенной связи необходимо определить признак зависимости по идентификации, т. е. выделить идентифицирующие и неидентифицирующие связи. Также выбрать способ реализации связей типа «Категория» и «Иерархическая рекурсия».

Неопределенные связи между сущностями необходимо разрешить за счет введения новых сущностей и разделения каждой неопределенной связи на две определенные идентифицирующие связи между двумя сущностями и третьей новой сущностью.

По итогам второго этапа проектирования строится КВ-диаграмма предметной области с указанием всех первичных и внешних ключей сущностей (рис. 4.16).

Для каждой сущности в качестве первичного ключа был выбран числовой атрибут «Номер». Для того чтобы не путаться при создании связей и миграции ключей, к каждому первичному ключу в конце названия добавлена буква от названия сущности: для сущности «Покупатель» первичным ключом является атрибут «НомерП», для сущности «Товар» – «НомерТ», для сущности «Накладная» – «НомерН».

Далее были реализованы выявленные на первом этапе связи. Определенная неидентифицирующая связь была реализована посредством миграции первичного ключа родительской сущности «Покупатель» в область атрибутов дочерней сущности «Накладная». Таким образом, в дальнейшем в каждой накладной будет указан покупатель, на которого она оформлена.

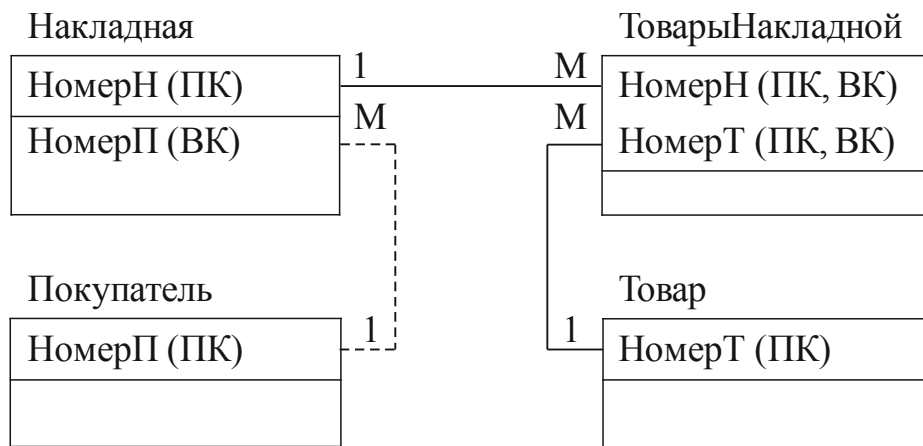


Рис. 4.16 – КВ-диаграмма Торговой компании

Неопределенная связь между сущностями «Накладная» и «Товар» была разрешена с помощью добавления новой сущности, которая получила название «ТоварыНакладной». Также были образованы две определенные идентифицирующие связи между новой сущностью и двумя старыми, при этом первичные ключи обеих родительских сущностей, мигрировавшие в дочернюю, образовали новый составной ключ.

Этап 3. Построение полноатрибутной модели

Заключительным этапом проектирования является создание итоговой полноатрибутной модели предметной области на основе полученной ранее КВ-диаграммы. ФА-диаграмма должна включать в себя, помимо сущностей, связей, первичных и внешних ключей, ещё и набор всех неключевых атрибутов каждой сущности, который позволит хранить весь необходимый объем информации о предметной области.

Для торговой компании итоговая диаграмма модели предметной области, изображенная на рисунке 4.17, дополнена набором неключевых атрибутов. Сущность «Покупатель» дополнена атрибутами «ФИО», «Адрес» и «Телефон». Сущность «Товары» теперь будет состоять из набора товаров, у каждого из которых есть наименование, категория и цена. В накладной будет указываться номер, покупатель и дата оформления, а также необходимость в доставке товаров. Список товаров каждой накладной будет храниться в сущности «ТоварыНакладной». Можно обратить внимание, что атрибут «Цена» встречается в двух сущностях одновременно – «Товар» и «ТоварыНакладной». Это связано с тем, что каждый товар имеет некоторую текущую цену, т. е. цену, по которой товар продается в данный момент. Эта информация хранится в сущности «Товар». Естественно, что эта цена может измениться со временем, т. е. цена одного и

того же товара в разных накладных, выписанных в разное время, может быть различной. Таким образом и возникло два атрибута «Цена» – цена товара в накладной и текущая цена товара.

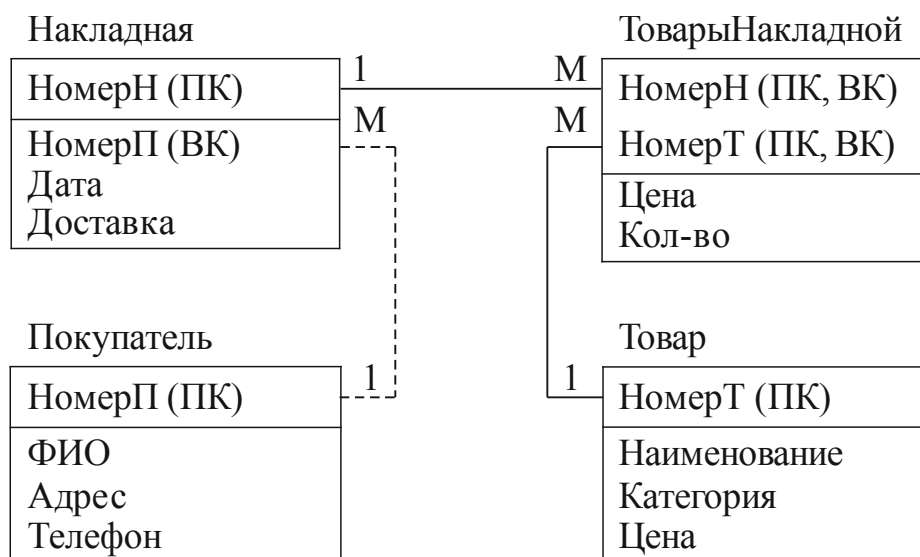


Рис. 4.17 – FA-диаграмма Торговой компании

4.3.4 CASE-средства

Существует целый ряд программных продуктов, предназначенных для автоматизации процесса проектирования, в том числе для построения диаграмм модели IDEFIX, а также других известных нотаций [14]. Сэкономить временные ресурсы и создать структуру БД помогают так называемые CASE-средства (Computer Aided Software Engineering) автоматизированного проектирования.



Под CASE-средствами понимается все программное обеспечение, используемое для автоматизации процесса разработки информационных систем.

CASE-средства могут охватывать как отдельные этапы жизненного цикла информационной системы, так и несколько этапов сразу.

Широкому внедрению CASE-средств в процесс разработки информационных систем способствовало:

- развитие методологии проектирования и формализация некоторых его процессов;
- подготовка профессиональных аналитиков и программистов, восприимчивых к структурному и объектно-ориентированному подходам в разработке ИС;

- рост производительности компьютеров;
- внедрение и широкое распространение сетевых технологий, позволяющих объединять усилия нескольких разработчиков в общий процесс проектирования.

В общем случае современные CASE-системы имеют в своем составе следующие компоненты:

1. Репозиторий, или словарь данных. Представляет собой специализированную базу данных, предназначенную для хранения состояний разрабатываемой информационной системы. В репозитории хранится исчерпывающая информация о разрабатываемой системе, в том числе и версии отдельных ее компонентов. Так что в любой момент проектировщики в случае неудачи могут вернуться к предыдущей версии.
2. Большое значение в использовании CASE-инструментов имеет развитие диаграммных графических техник, позволяющих моделировать поведение предметной области в графическом виде. По этой причине все основные CASE-средства имеют в своем составе специализированные *графические редакторы*.
3. Верификатор диаграмм. Служит для проверки построенных в процессе разработки диаграмм на предмет соответствия используемой методологии.
4. Документатор проекта. Позволяет получать информацию о состоянии разработки проекта. С помощью документатора можно автоматизировать процесс создания документации.
5. Администратор проекта. Предназначен:
 - для инициализации проекта;
 - задания начальных параметров;
 - назначения и изменения прав доступа к элементам проекта;
 - мониторинга выполнения проекта.
6. Сервисы. Предназначены для выполнения различных вспомогательных функций, например архивирования или восстановления проекта из архива.

Классификация CASE-средств

Существует несколько подходов к классификации CASE-инструментов. Рассмотрим классификацию, ориентированную на процессы жизненного цикла информационных систем.

- *Средства анализа и проектирования*: BPWin, Silverrun, Oracle Designer, Rational Rose, Paradigm Plus, Power Designer, System Architect.
- *Средства проектирования баз данных*: ERWin, Silverrun, Oracle Designer, Paradigm Plus, Power Designer, MySQL Workbench. Следует также отметить, что практически все СУБД имеют в своем составе средства графического проектирования баз данных.
- *Средства управления требованиями*: RequisitePro, DOORS (Dynamic Object-Oriented Requirements System, динамическая объектно-ориентированная система управления требованиями).
- *Средства управления конфигурацией*: PVCS, ClearCase, Telelogic Synergy.
- *Средства документирования*: Rational SoDA (Software Document Automation, автоматизированное документирование программного обеспечения).
- *Средства тестирования*: Rational Suite TestStudio, Rational Robot.
- *Средства управления проектами*: Open Plan Professional, Microsoft Project, Borland StarTeam, IBN.
- *Средства переноса программного обеспечения в новую среду*: Rational Rose, Paradigm Plus [15].

В качестве примера рассмотрим простейшее CASE-средство, которое позволяет осуществлять частичную автоматизацию процесса проектирования БД MySQL. В программном продукте MySQL Workbench разработчик имеет возможность проектировать модель предметной области с помощью редактора ER-диаграмм и в дальнейшем интегрировать полученную модель непосредственно в СУБД MySQL.

На рисунке 4.18 приведена модель предметной области «Торговая компания», реализованная в MySQL Workbench с использованием нотации IDEF1X.

На рисунке 4.19 приведена та же самая диаграмма с выбором нотации IE для изображения связей.

MySQL Workbench является простейшим бесплатным CASE-средством и не позволяет напрямую взаимодействовать с СУБД, отличными от MySQL. Тем не менее данный продукт, как и большинство других CASE-средств, содержит встроенный генератор SQL-кода, который на основе графического представления предметной области генерирует код, позволяющий реализовать данную модель в виде готовой БД в произвольной СУБД. Для представленной на ри-

сунке 4.19 модели данных сгенерированный SQL-код приведен в приложении А.

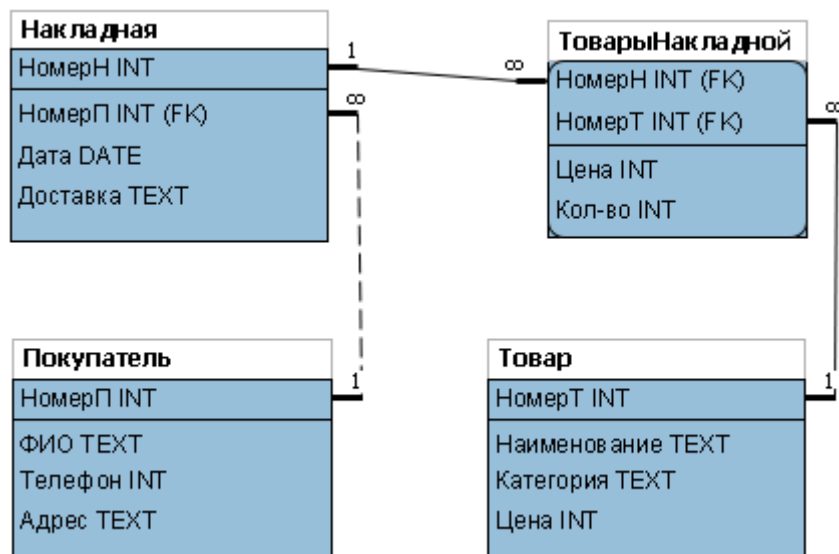


Рис. 4.18 – Пример ER-диаграммы в нотации IDEF1X

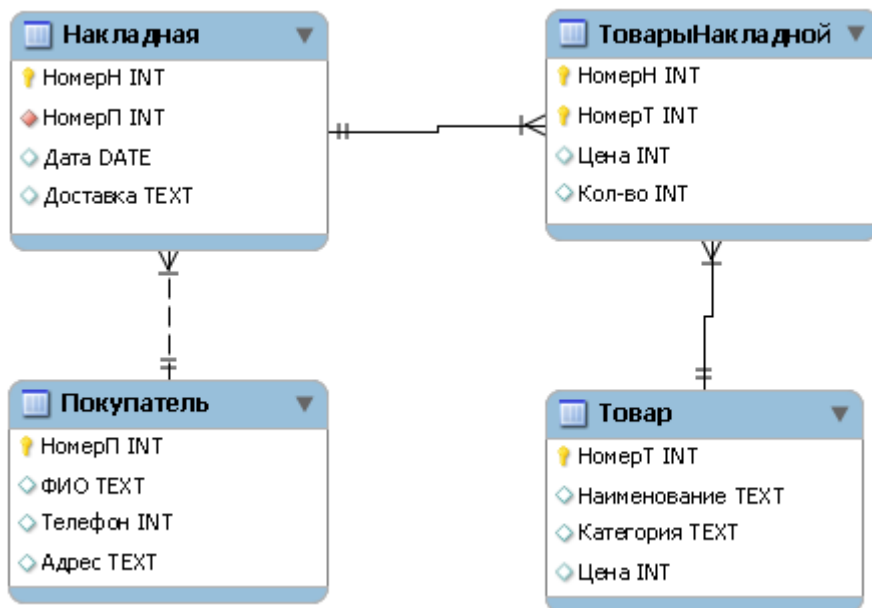


Рис. 4.19 – Пример ER-диаграммы в нотации IE

Использование специализированных систем проектирования позволяет разработчику оптимизировать действия, связанные с проектированием, а также частично автоматизировать процесс проектирования предметной области и в дальнейшем наиболее комфортным образом преобразовать полученную модель в готовую базу данных.



Контрольные вопросы по главе 4

1. Перечислите основные этапы жизненного цикла БД.
2. Что такое аномалии и какие бывают виды аномалий?
3. Назовите условия нахождения таблицы во второй нормальной форме.
4. В какой нормальной форме находится по умолчанию реляционная таблица с простым первичным ключом?
5. С какой целью при проектировании БД применяется ER-модель?
6. В какую область дочерней сущности мигрирует первичный ключ родительской сущности при установлении идентифицирующей связи?
7. Дайте определение термину CASE-средства.

5 Администрирование баз данных

Для успешного функционирования информационной системы, использующей БД, недостаточно выбора СУБД и сервера БД. На начальной стадии запуска информационной системы и в процессе ее эксплуатации необходимо выполнять настройку и различные функции администрирования. Важнейшими задачами администрирования являются обеспечение защиты информации в БД и бесперебойного функционирования СУБД [3].

Информация, содержащаяся в БД, должна обладать следующими свойствами:

- доступность – возможность за приемлемое время выполнить ту или иную операцию над данными или получить нужную информацию. Например, защита данных от повреждения является лишь частным случаем защиты от нарушения доступности информации;
- целостность – это актуальность и непротиворечивость хранимой информации. Актуальность следует понимать как оперативное отражение изменений, происходящих в предметной области, в БД. Непротиворечивость информации – это соответствие содержимого БД логике предметной области;
- конфиденциальность – защищенность информации от несанкционированного доступа.

Для обеспечения этих свойств в рамках СУБД перед администратором стоит целый ряд задач настройки и управления:

- резервное копирование данных;
- разграничение прав доступа к данным;
- ведение журналов и аудит;
- контроль ввода данных;
- периодическое тестирование системы;
- шифрование данных и др.

Рассмотрим подробнее виды угроз для информации в БД и различные методы администрирования СУБД для предотвращения проблем её функционирования.

5.1 Безопасность БД



Безопасность БД – это защищенность информации и поддерживающей инфраструктуры от случайных или преднамеренных воздействий естественного или искусственного характера, которые могут нарушить доступность, целостность или конфиденциальность информации в БД.

Угроза безопасности БД – действие или событие, которое может привести к нарушению доступности, достоверности, целостности или конфиденциальности информации в БД.

Классификация угроз

Выделим несколько критериев, по которым будет произведена классификация угроз.

1. Если говорить об *объекте* угроз, то можно рассматривать следующие виды:

- угроза непосредственно данным, хранящимся в БД;
- угроза связанному с использованием данных программному обеспечению, например СУБД или информационной системе, работающей с данными;
- угроза системному программному обеспечению;
- угроза компьютерной технике и связанному с ней оборудованию и т. д.

2. По *характеру* все угрозы информационной безопасности делятся на *случайные* и *преднамеренные*. Случайные угрозы возникают независимо от воли и желания людей, хотя люди и могут быть источниками этих угроз. Преднамеренные же угрозы всегда создаются людьми осознанно.

3. Ещё одним критерием угроз безопасности является *источник происхождения* угрозы:

3.1. *Природные угрозы*. Природные угрозы являются случайными и связаны прежде всего с прямым физическим воздействием на компьютерную систему различных природных явлений и катаклизмов.

3.2. *Технические угрозы*. Это угрозы, вызванные неполадками аппаратных средств (компьютеров, сетевого или электрического оборудования и др.) или программных средств (информационной системы, операционные системы, драйверы, сетевое программное обеспечение и др.). К техническим же угрозам

следует отнести и неполадки в системах жизнеобеспечения (здания, тепло-, водоснабжение и т. д.).

3.3. *Угрозы, созданные людьми.* Угрозы, созданные людьми, можно разделить:

- на угрозы со стороны людей, работающих с БД (обслуживающий персонал, программисты, администраторы, операторы, управленческий аппарат и др.);
- угрозы, вызванные внешними злоумышленниками, в том числе хакерские атаки, компьютерные вирусы и др.

4. Кроме перечисленных выше признаков, по которым можно классифицировать угрозы информационной безопасности, угрозы можно поделить на *прямые* и *косвенные*.

Косвенные угрозы непосредственно не приводят к каким-либо нежелательным явлениям в компьютерной системе, но они могут оказаться источниками новых косвенных или прямых угроз. Например, запись на диск зараженного компьютерным вирусом файла может лишь с некоторой вероятностью привести к неполадкам в работе операционной системы. В свою очередь неполадки в операционной системе также с некоторой вероятностью приведут к неполадкам в информационной системе. Знание возможных косвенных угроз помогает просчитать непосредственные угрозы для информационной системы и повысить уровень безопасности.

5. Все угрозы для *информации*, хранящейся в БД, делятся на три класса:

- угрозы доступности информации;
- угрозы целостности информации;
- угрозы конфиденциальности информации.

Угрозы доступности информации в БД

К угрозам доступности информации в БД относятся те события или проблемы, которые приводят к состоянию, когда информация из БД становится полностью или частично недоступной для пользователей или программ.

1. Ошибки пользователей БД.

Под пользователями можно понимать различные группы людей (администраторы, программисты, проектировщики, пользователи), работающие с информационной системой, а следовательно и с БД. Непреднамеренные ошибки пользователей могут вызвать непосредственно порчу данных или средств доступа либо создать условия для реализации других угроз.

2. Отказы информационной системы.

Источниками отказа информационной системы могут быть следующие:

- выход системы из штатного режима эксплуатации в силу случайных или преднамеренных действий пользователей;
- отказ программного обеспечения, содержащего различные ошибки, которые могут привести к повреждениям данных;
- ошибки конфигурирования системы. В информационных системах конфигурирование выполняется при установке и настройке, и в случае допущенных ошибок могут возникнуть проблемы в эксплуатации;
- повреждение самих данных вследствие аппаратных или программных ошибок.

3. Внешние источники угрозы доступности БД.

Источниками внешних угроз могут быть следующие:

- отказ, повреждение или разрушение аппаратных средств (носителей информации, компьютеров, сетевого оборудования и т. д.);
- нарушение условий работы (системы связи, электропитание, отопление и т. п.). Например, внезапное отключение электричества может служить серьезной проблемой функционирования любых компьютерных систем;
- сетевые атаки, вирусные программы и различное вредоносное программное обеспечение;
- разрушение информации намеренными действиями человека, не относящегося к категории пользователей системы.

Угрозы целостности информации в БД

Нарушение целостности БД может произойти по причинам, по которым может произойти и нарушение доступности БД. Например, ошибка пользователя может привести к тому, что данные перестанут отражать реально существующие отношения в предметной области. Аналогично к нарушению целостности могут привести и внутренние отказы ИС, и любые внешние случайные или намеренные воздействия. В свою очередь нарушение целостности может в конце концов привести и к нарушению доступа к данным. Например, если в базе данных содержится информация о платежах, которые произвела компания, а номер месяца платежа окажется больше 12, то следствие может быть каким угодно, начиная от выдачи неверных данных и заканчивая сбоем в работе всей

системы. Все зависит от того, какие алгоритмы обработки данных используются в информационной системе.

Угрозы конфиденциальности информации в БД

Угрозы конфиденциальности информации в БД связаны с несанкционированным доступом к данным, который может быть получен в результате следующих ситуаций:

- ошибки в настройке или полное отсутствие систем ограничения доступа;
- сбой или отказ программного или аппаратного обеспечения;
- последствия ошибок пользователей;
- использование специального (шпионского) программного обеспечения и аппаратных средств.

Система защиты БД должна свести к минимуму все перечисленные угрозы конфиденциальности. Далее рассмотрим возможные методы администрирования и настройки баз данных, которые позволяют избежать угроз безопасности информации в БД [15].

5.2 Резервное копирование БД

После того как предприятие принимает решение о хранении и обработке своих данных с помощью системы управления базой данных, оно становится полностью зависимым от бесперебойного функционирования этой системы. В случае повреждения какой-либо части базы данных вследствие ошибки человека, отказа оборудования или сбоя программ операционной системы очень важно иметь возможность восстановить утраченные данные с минимальной задержкой и с наименьшим воздействием на остальную часть системы. Администратор БД должен разработать и реализовать подходящую схему восстановления, которая может включать в себя периодическую выгрузку базы данных на устройство резервного копирования, процедуры загрузки базы данных из последней созданной копии в случае необходимости, инструменты отслеживания текущего состояния БД и т. д. Зачастую большинство инструментов резервного копирования и восстановления данных содержит в себе непосредственно сама СУБД. С другой стороны, не редки случаи, когда многие из инструментов восстановления приходится реализовывать либо самостоятельно, либо с использованием дополнительного программного обеспечения и утилит [6].

Резервное копирование означает периодическое сохранение файлов БД на внешнем запоминающем устройстве. Оно выполняется тогда, когда состояние файлов БД является непротиворечивым. Резервная копия (РК) не должна создаваться на том же диске, на котором находится сама БД, т. к. при аварии диска базу невозможно будет восстановить. В случае сбоя (или аварии диска) БД восстанавливается на основе последней копии.

Полная резервная копия включает всю базу данных (все файлы БД, в том числе вспомогательные, состав которых зависит от СУБД). Частичная резервная копия включает часть БД, определённую пользователем. Резервная копия может быть инкрементной, т. е. состоять только из тех блоков (страниц памяти), которые изменились со времени последнего резервного копирования. Создание инкрементной копии происходит быстрее, чем полной, но оно возможно только после создания полной резервной копии.

Создание частичной и инкрементной РК выполняется средствами СУБД, а создание полной РК – средствами СУБД или операционной системы. В резервную копию, созданную средствами СУБД, обычно включаются только те блоки памяти, которые реально содержат данные (т. е. пустые блоки, выделенные под объекты БД, в резервную копию не входят).

Периодичность резервного копирования определяется администратором системы и зависит от многих факторов: объём БД, интенсивность запросов к БД, интенсивность обновления данных и др. Как правило, технология проведения резервного копирования такова:

- раз в неделю (день, месяц) осуществляется полное копирование;
- раз в день (час, неделю) – частичное или инкрементное копирование.

Все изменения, произведённые в данных после последнего резервного копирования, утрачиваются; но при наличии *архива журнала транзакций* их можно выполнить ещё раз, обеспечив полное восстановление БД на момент возникновения сбоя. Дело в том, что журнал транзакций содержит сведения только о текущих транзакциях. После завершения транзакции информация о ней может быть перезаписана. Для того чтобы в случае сбоя обеспечить возможность полного восстановления БД, необходимо вести архив журнала транзакций, т. е. сохранять копии файлов журнала транзакций вместе с резервной копией базы данных.

5.3 Настройка доступа к БД

Для защиты БД от несанкционированного доступа прежде всего необходима эффективная система регистрации попыток доступа в систему со стороны пользователей и программ, а также мгновенная сигнализация о них администраторам БД. Чтобы регистрировать события подключения к системе, в БД обычно ведется специальный журнал или база данных.

Защита от несанкционированного доступа со стороны пользователей в современных системах в основном реализуется с помощью парольной защиты либо в более развитых системах на основе методов идентификации и аутентификации.

Парольная защита является простейшим средством защиты от несанкционированного доступа. Основной ее недостаток состоит в том, что многие пользователи, использующие одинаковый пароль, с точки зрения БД неразличимы. При небрежном отношении к записям пароль может стать достоянием других. После ввода этого пароля обычно разрешается большинство операций над данными. Иногда в системе имеется несколько паролей, за каждым из которых закреплены соответствующие права доступа.

Более серьезный контроль доступа в систему можно организовать, если каждого подключающегося пользователя сначала идентифицировать, затем убедиться, что это именно он, а не другой (аутентифицировать), и при запросе ресурсов контролировать полномочия (проверять право запрашивать ресурсы системы). Большинство СУБД имеют лишь средства идентификации пользователей, тогда как наиболее развитые и современные включают дополнительные средства аутентификации.



***Идентификация** – это передача субъектом (пользователем или процессом) системе своего имени (идентификатора). На этапе **аутентификации** происходит проверка подлинности переданного имени.*

Идентификация пользователей может выполняться, например, с помощью ввода имени пользователя и пароля. Для аутентификации, или проверки подлинности пользователя, часто используют следующие способы:

- запрос секретного пароля;
- запрос какой-либо информации сугубо индивидуального характера;

- проверка наличия физического объекта, представляющего собой электронный аналог обычного ключа (электронный ключ);
- биометрические средства.

Запрашиваемая для аутентификации дополнительная информация может представлять собой произвольные данные, связанные с некоторыми сведениями, явлениями или событиями из личной жизни пользователя или его родственников. Например, номер счета в банке, номер технического паспорта автомобиля, девичья фамилия матери или жены и т. д.

Примером электронного ключа может быть пластиковая карточка с магнитной полоской или устройство в виде флешки.

Из множества существующих средств аутентификации наиболее надежными считаются биометрические средства. В них опознание личности осуществляется по отпечаткам пальцев, форме ладони, сетчатке глаза, подписи, голосу и другим физиологическим параметрам человека. Некоторые системы идентифицируют человека по манере работы на клавиатуре. Основным достоинством систем такого класса является высокая надежность аутентификации. Недостатки систем включают в себя повышенную стоимость оборудования, временную задержку распознавания некоторых средств и неудобство для пользователя [3].

Уверенность в том, что подключающийся к системе пользователь или программа не являются злоумышленниками, не дает гарантии безопасности последующего поведения во время работы, поэтому во многих системах защиты предусматривается разграничение доступа к объектам и ресурсам в течение сеанса работы с БД.

Разграничение доступа не только позволяет сохранять конфиденциальность информации, но и защищает ее от угроз доступности и целостности. Правильное распределение ролей в системе может уберечь объекты безопасности от случайных или некомпетентных действий, которые могут вызвать потерю или искажение информации.

В общем случае постановка задачи следующая. Имеется множество субъектов безопасности (пользователи или процессы), доступ которых к информации в БД предполагается регулировать. С другой стороны, имеется множество объектов данных, доступ к которым субъектов безопасности может регулироваться. В качестве объектов могут выступать самые разные элементы, такие как базы данных, таблицы, столбцы, строки и т. д. Для каждой пары «субъект – объект» должно быть определено множество операций, которые субъект может

выполнять над объектом. Операции могут быть простыми, такими как добавление данных, удаление данных, обновление данных, а могут представлять собой хранимые процедуры, содержащие множество различных действий.

Кроме ограничений на действия с данными для субъектов безопасности субъекту могут даваться разрешения на операции выдачи прав или введение ограничений на действия с данными других субъектов. Это особо тонкая работа, требующая вдумчивого подхода. Достаточно случайно дать дополнительные права какому-либо пользователю, и он сам сможет расширить свои права для работы с данными или создать нового пользователя с расширенными правами.

Для упрощения работы с большим количеством пользователей используется понятие роли (или группы пользователей). На пользователя, являющегося членом данной роли, распространяются все разрешения или запреты, присвоенные данной роли. Таким образом, для того чтобы дать разрешение на выполнения каких-либо операций над данными для большого количества пользователей, достаточно объединить их в одной роли, а затем разрешить эти действия для роли. Обычно роли создают для выполнения каких-либо сходных действий. Например, можно создать роль «Бухгалтер» и присоединить к ней всех бухгалтеров предприятия. После этого можно легко управлять целой группой пользователей [15].

5.4 Дополнительные инструменты защиты БД

К дополнительным средствам защиты БД можно отнести целый ряд инструментов, доступных для администраторов и программистов в различных СУБД: шифрование, средства контроля вводимых значений, протоколирование и аудит, тестирование различных аспектов безопасности БД и т. д.

1. Шифрование.

Одним из наиболее мощных средств обеспечения конфиденциальности данных является *шифрование*. Шифрование данных (всей базы или отдельных таблиц) применяют для того, чтобы в случае попадания файлов базы данных к злоумышленнику процесс извлечения информации из файлов был крайне затруднителен. Кроме шифрования файлов БД иногда применяется шифрование исходных текстов программ, которое позволяет скрыть от несанкционированного пользователя описание соответствующих алгоритмов. Шифрование также целесообразно использовать в системах, передающих данные по от-

крытым каналам связи, с целью невозможности извлечения информации из перехваченных данных.

2. Средства контроля вводимых значений.

Во многих СУБД для предотвращения случайных ситуаций ввода некорректных данных используются специальные средства контроля. Редактируя БД, пользователь может случайно ввести такие значения, которые не соответствуют типу поля, в которое это значение вводится. Например, в числовое поле пользователь пытается занести текстовую информацию. В этом случае СУБД с помощью *средств контроля значений* блокирует ввод и сообщает пользователю об ошибке звуковым сигналом, изменением цвета вводимых символов или другим способом.

Средства повышения достоверности вводимых значений в СУБД служат для более глубокого контроля, связанного с семантикой обрабатываемых данных. Они обычно обеспечивают возможность при создании таблицы указывать следующие ограничения на значения: минимальное и максимальное значения; значение, принимаемое по умолчанию (если нет ввода), требование обязательного ввода; задание маски (шаблона) ввода; указание дополнительной сверочной таблицы, по которой ведется контроль вводимых значений и т. д.

Более совершенной формой организации контроля достоверности информации в БД является разработка хранимых процедур. Механизм хранимых процедур применяется в БД, размещенных на сервере. Сами хранимые процедуры представляют собой программы, алгоритмы которых предусматривают выполнение некоторых функций (в том числе контрольных) над данными. Процедуры хранятся вместе с данными и при необходимости вызываются из приложений либо при наступлении некоторых событий в БД [3].

3. Протоколирование и аудит.

Зачастую помогает обеспечить защищенность данных наличие в системе *протоколирования* (запись в журнал) всех действий, которые совершают пользователи. Администратор может отслеживать все действия пользователей и выявлять возникающие на их основе проблемы. Кроме протоколирования для анализа накопленных данных используется аудит. Аудит – это периодический или в реальном времени анализ накопленных данных по активности пользователей. Аудит может выявить нештатные ситуации и сигнализировать о них администраторам системы. Под нештатными ситуациями понимается целый набор событий, которые могут свидетельствовать о возможных угрозах системе (сетевые атаки, попытки подобрать пароль и т. п.).

4. *Тестирование системы.*

Необходимо периодически проводить тщательное тестирование информационной системы и базы данных на предмет уязвимостей и на предмет критических ситуаций (при большой нагрузке, больших объемах обрабатываемой информации и т. п.), а также на предмет ввода заведомо неправильной информации [15].



Контрольные вопросы по главе 5

1. Дайте определение термину «безопасность БД».
2. Перечислите основные критерии классификации угроз БД.
3. Назовите угрозы конфиденциальности информации.
4. Что такое идентификация?
5. Перечислите основные способы аутентификации.

Заключение

Базы данных являются основой большинства действующих информационных систем, а информация, накопленная в различных базах данных, лежит в основе развития многих процессов в современном обществе. Таким образом, важность и значимость инструментов управления базами данных определяют серьезные требования к квалификации специалистов, занимающихся проектированием, управлением, а также взаимодействием с информационными системами в любой предметной области. При возрастающем объеме информации и сложности операций работы с данными проблема понимания эффективного управления хранением, доступом и обработкой информации становится всё более значимой.

Изучив пособие, вы освоите основные принципы функционирования и проектирования баз данных, которые могут позволить в дальнейшем применить её в рамках информационной системы.

Литература

1. Информатика. Базовый курс / под ред. С. В. Симоновича. – 2-е изд. – СПб. : Питер, 2005. – 640 с.: ил.
2. Глушаков С. В. Базы данных : учеб. курс / С. В. Глушаков, Д. В. Ломотько. – Харьков : Фолио; М. : ООО «Издательство АСТ», 2002. – 504 с.
3. Хомоненко А. Д. Базы данных : учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев ; под ред. проф. А. Д. Хомоненко. – 4-е изд., доп. и перераб. – СПб. : КОРОНА принт, 2004. – 736 с.
4. Гарсиа-Молина Г. Системы баз данных. Полный курс / Г. Гарсиа-Молина, Дж. Ульман, Дж. Уидом. – Б. м. : Вильямс, 2003. – 1088 с.
5. Марков А. С. Базы данных. Введение в теорию и методологию : учебник / А. С. Марков, К. Ю. Лисовский. – М. : Финансы и статистика, 2006. – 512 с.: ил.
6. Дейт К. Дж. Введение в системы баз данных / К. Дж. Дейт ; пер. с англ. – 8-е изд. – М. : Вильямс, 2005. – 1328 с.
7. Кузин А. В. Базы данных : учеб. пособие для студ. высш. учеб. заведений / А. В. Кузин, С. В. Левонисова. – 2-е изд., стер. – М. : ИЦ «Академия», 2008. – 320 с.
8. Карпова Т. С. Базы данных: модели, разработка, реализация / Т. С. Карпова – СПб. : Питер, 2002. – 304 с.
9. Бекаревич Ю. Б. Самоучитель Access 2010 / Ю. Б. Бекаревич, Н. В. Пушкина – СПб. : БХВ-Петербург, 2011. – 432 с.: ил.
10. Кузнецов М. В. MySQL 5 / М. В. Кузнецов, И. В. Симдянов. – СПб. : БХВ-Петербург, 2010. – 1024 с.: ил.
11. Реализация языка SQL в MySQL [Электронный ресурс]. – Режим доступа: http://www.sql.ru/docs/mysql/rus_ref/sql.shtml (дата обращения: 14.12.2017).
12. Дунаев В. В. Базы данных. Язык SQL / В. В. Дунаев. – СПб. : БХВ-Петербург, 2006. – 288 с.
13. Гвоздева Т. В. Проектирование информационных систем / Т. В. Гвоздева, Б. А. Баллод. – Ростов н/Д : Феникс, 2009. – 512 с.

14. Боровской И. Г. Специализированная подготовка разработчиков бизнес-приложений : учеб. пособие / И. Г. Боровской, С. И. Колесникова, А. А. Матолыгин. – Томск : ТУСУР, 2007. – 267 с.
15. Пирогов В. Ю. Информационные системы и базы данных: организация и проектирование : учеб. пособие / В. Ю. Пирогов. – СПб. : БХВ-Петербург, 2009. – 528 с.: ил.

Список условных обозначений и сокращений

1:1 – связь «ОДИН-К-ОДНОМУ»

1:M – связь «ОДИН-КО-МНОГИМ»

1НФ – первая нормальная форма

2НФ – вторая нормальная форма

3НФ – третья нормальная форма

4НФ – четвертая нормальная форма

5НФ – пятая нормальная форма

AL – Administration Language

ANSI – American National Standard Institute

API – Application Programming Interface

CASE – Computer Aided Software Engineering

CODASYL – Conference of Data System Languages

DB – Database

DBMS – Database Management System

DDL – Data Definition Language

DML – Data Manipulation Language

DOORS – Dynamic Object-Oriented Requirements System

DQL – Data Query Language

ER – Entity Relationship

FA – Full Attributed

FK – Foreign Key

GPL – General Public License

IE – Information Engineering

ISO – International Standards Organization

KB – Key Based

M:N – связь «МНОГИЕ-КО-МНОГИМ»

ODMG – Object Database Management Group

OLAP – OnLine Analytical Processing

PK – Primary Key

SoDA – Software Document Automation

SPARC – Standards Planning And Requirements Committee

SQL – Structured Query Language

TCL – Transaction Control Language

VBA – Visual Basic for Applications

АИС – автоматизированная информационная система

БД – база данных

ВК – внешний ключ

ЖЦ – жизненный цикл

ИС – информационная система

НФБК – нормальная форма Бойса – Кодда

ПК – первичный ключ

ПО – предметная область

РК – резервная копия

РМД – реляционная модель данных

СУБД – система управления базами данных

ФИО – фамилия, имя, отчество

ЭВМ – электронная вычислительная машина

Глоссарий

CASE-средства – программное обеспечение, используемое для автоматизации процесса разработки информационных систем.

Аномалия БД – ситуация в таблицах БД, приводящая к противоречиям и усложняющая процессы обработки данных.

Атрибут – одно из свойств, характеризующих объект (сущность).

База данных – поименованная совокупность взаимосвязанных данных, отображающая состояние объектов и их связей в некоторой предметной области и находящаяся под управлением специального программного комплекса.

Безопасность БД – это защищенность информации и поддерживающей инфраструктуры от случайных или преднамеренных воздействий естественного или искусственного характера, которые могут нарушить доступность, целостность или конфиденциальность информации в БД.

Внешний ключ – это атрибут или набор атрибутов отношения, значения которого являются значениями первичного ключа другого отношения, связанного с этим.

Данные – это сведения об объектах окружающего мира, введенные на специальный носитель и предназначенные для хранения, передачи и обработки.

Домен – множество всех возможных значений атрибута отношения.

Жизненный цикл баз данных – это непрерывный процесс, начинающийся с момента принятия решения о необходимости создания базы данных и заканчивающийся в момент полного ее изъятия из эксплуатации.

Инфологическое проектирование – проектирование инфологической модели предметной области, т. е. формализованного описания объектов предметной области в терминах некоторой семантической модели, например в терминах модели «сущность – связь», или ER-модели.

Информационная система – программный комплекс, реализованный для сбора, хранения и обработки информации для принятия решений и реализации функций управления.

Информация – это совокупность сведений о фактических данных и зависимостях между ними.

Кортеж – упорядоченный набор значений атрибутов.

Логическое проектирование – процесс разработки корректной схемы базы данных в терминах выбранной СУБД.

Модель данных – это совокупность структур данных и операций над этими структурами, поддерживаемых конкретной СУБД.

Мощность связи – количество экземпляров дочерней сущности, которое может существовать для каждого экземпляра родительской сущности.

Неопределенная связь – отношение между двумя сущностями, при котором каждый экземпляр первой сущности связан с 0, 1 или более экземплярами второй сущности и каждый экземпляр второй сущности связан с 0, 1 или более экземплярами первой сущности.

Нормализация – процесс реорганизации данных путем устранения повторяющихся групп и других противоречий в организации данных с целью приведения набора таблиц к виду, позволяющему осуществлять корректные операции над данными.

Определенная связь – отношение между сущностями, в котором каждый экземпляр родительской сущности связан с 0, 1 или более экземплярами дочерней сущности и каждый экземпляр дочерней сущности связан с 0 или 1 экземпляром родительской сущности.

Отношение – двумерная таблица, содержащая некоторые данные о предметной области.

Первичный ключ – атрибут или набор атрибутов отношения, однозначно идентифицирующий любой из его кортежей.

Распределенная БД – это набор логически связанных между собой разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети, но воспринимаются пользователями как единая БД.

Распределенная СУБД – это программный комплекс, предназначенный для управления распределенными БД и позволяющий сделать распределённость информации прозрачной для конечного пользователя.

Реляционная модель данных – совокупность взаимосвязанных отношений, изменяющихся во времени и хранящая сведения о некоторой предметной области.

Связь – ассоциация между сущностями, которая показывает, каким образом сущности взаимодействуют или соотносятся между собой.

Система управления базами данных – это совокупность программных и языковых средств, предназначенных для создания, ведения и совместного использования базы данных многими пользователями и обеспечения её взаимодействия с прикладными программами.

Сущность – произвольный объект реального мира, данные о котором необходимо хранить в базе данных.

Угроза безопасности БД – действие или событие, которое может привести к нарушению доступности, достоверности, целостности или конфиденциальности информации в БД.

Приложение А SQL-код для создания базы данных

```
CREATE DATABASE IF NOT EXISTS `mydb`;
```

```
USE `mydb` ;
```

```
-- -----  
-- Table `mydb`.`Покупатель`  
-- -----
```

```
DROP TABLE IF EXISTS `mydb`.`Покупатель` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Покупатель` (
```

```
  `НомерП` INT NOT NULL ,
```

```
  `ФИО` TEXT NULL ,
```

```
  `Телефон` INT NULL ,
```

```
  `Адрес` TEXT NULL ,
```

```
  PRIMARY KEY (`НомерП`) )
```

```
ENGINE = InnoDB;
```

```
-- -----  
-- Table `mydb`.`Накладная`  
-- -----
```

```
DROP TABLE IF EXISTS `mydb`.`Накладная` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Накладная` (
```

```
  `НомерН` INT NOT NULL ,
```

```
  `НомерП` INT NOT NULL ,
```

```
  `Дата` DATE NULL ,
```

```
  `Доставка` TEXT NULL ,
```

```
  PRIMARY KEY (`НомерН`) ,
```

```
  INDEX `fk_Накладная_Покупатель1` (`НомерП` ASC) ,
```

```
  CONSTRAINT `fk_Накладная_Покупатель1`
```

```
  FOREIGN KEY (`НомерП` )
```

```
  REFERENCES `mydb`.`Покупатель` (`НомерП` )
```

```
  ON DELETE NO ACTION
```

```
  ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

```
-- -----  
-- Table `mydb`.`Товар`  
-- -----
```

```

DROP TABLE IF EXISTS `mydb`.`Товар` ;
CREATE TABLE IF NOT EXISTS `mydb`.`Товар` (
  `НомерТ` INT NOT NULL ,
  `Наименование` TEXT NULL ,
  `Категория` TEXT NULL ,
  `Цена` INT NULL ,
  PRIMARY KEY (`НомерТ`) )
ENGINE = InnoDB;

-- -----
-- Table `mydb`.`ТоварыНакладной`
-- -----

DROP TABLE IF EXISTS `mydb`.`ТоварыНакладной` ;
CREATE TABLE IF NOT EXISTS `mydb`.`ТоварыНакладной` (
  `НомерН` INT NOT NULL ,
  `НомерТ` INT NOT NULL ,
  `Цена` INT NULL ,
  `Кол-во` INT NULL ,
  PRIMARY KEY (`НомерН`, `НомерТ`) ,
  INDEX `fk_Накладная_has_Товар_Накладная1` (`НомерН`
ASC) ,
  INDEX `fk_Накладная_has_Товар_Товар1` (`НомерТ`
ASC) ,
  CONSTRAINT `fk_Накладная_has_Товар_Накладная1`
FOREIGN KEY (`НомерН`)
REFERENCES `mydb`.`Накладная` (`НомерН`)
ON DELETE NO ACTION
ON UPDATE NO ACTION,
  CONSTRAINT `fk_Накладная_has_Товар_Товар1`
FOREIGN KEY (`НомерТ`)
REFERENCES `mydb`.`Товар` (`НомерТ`)
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB;

```