ЗРЮМОВ Е. А., ЗРЮМОВА А. Г.

БАЗЫ ДАННЫХ ДЛЯ ИНЖЕНЕРОВ

БАРНАУЛ 2010

Министерство образования и науки Российской Федерации

Федеральное агентство по образованию

ГОУ ВПО «Алтайский государственный технический университет
им. И. И. Ползунова»

Е. А. Зрюмов, А. Г. Зрюмова

БАЗЫ ДАННЫХ ДЛЯ ИНЖЕНЕРОВ

Учебное пособие

Рекомендовано Алтайским государственным техническим
университетом им. И. И. Ползунова в качестве учебного пособия
для студентов АлтГТУ, обучающихся по специальности
200106 «Информационно-измерительная техника и технологии»

Рекомендовано Учебно-методическим объединением вузов
Российской Федерации по образованию в области приборостроения и
оптотехники для студентов высших учебных заведений, обучающихся
по направлению подготовки 200100 «Приборостроение» и специальности
200106 «Информационно-измерительная техника и технологии»

Изд-во АлтГТУ
Барнаул • 2010

УДК 681.3.016(075.8)

Зрюмов, Е. А. Базы данных для инженеров [Текст] : учебное пособие / Е. А. Зрюмов, А. Г. Зрюмова; Алт. гос. техн. ун-т им. И. И. Ползунова. – Барнаул : Изд-во АлтГТУ, 2010. – 131 с.

ISBN 978-5-7568-0809-4

Учебное пособие содержит необходимые теоретические сведения для изучения принципов работы современных баз данных, а также практические реализации баз данных и приложений в Microsoft Office Access 2007, Microsoft Visual Studio с помощью технологии ADO.NET, MySQL с помощью языка программирования PHP. Учебное пособие направлено на формирование профессиональных компетенций инженеров, деятельность которых связана с информационными технологиями.

Рекомендовано Учебно-методическим объединением вузов Российской Федерации по образованию в области приборостроения и оптотехники для студентов высших учебных заведений, обучающихся по направлению подготовки 200100 «Приборостроение» и специальности 200106 «Информационно-измерительная техника и технологии». Протокол № 1 от 2 февраля 2010 г.

Рецензенты:

С. П. Пронин, д.т.н., профессор, заведующий кафедрой «Информационные технологии» Алтайского государственного технического университета им. И. И. Ползунова;

А. А. Веряев, д.п.н., профессор кафедры «Теоретические основы информатики» Алтайской государственной педагогической академии;

С. П. Пронь, к.т.н., доцент кафедры «Теоретическая кибернетика и прикладная математика» Алтайского государственного университета

ISBN 978-5-7568-0809-4

© Алтайский государственный технический университет им. И. И. Ползунова, 2010

© Е. А. Зрюмов, А. Г. Зрюмова, 2010

Содержание

Предисловие 6
Введение в базы данных 7

ГЛАВА 1
Основы проектирования баз данных 9
1.1 Терминология в СУБД 9
1.2 Поколения СУБД 10
1.3 Этапы проектирования баз данных 11
Контрольные вопросы 14
Список рекомендуемой литературы 14

ГЛАВА 2
Концепции разработки баз данных 15
2.1 Модель ANSI/SPARC 15
2.2 Физическая организация данных 16
2.3 Обмен информацией в базе данных 20
2.4 Этапы доступа к базе данных 22
2.5 Модели данных 23
2.5.1 Инфологические модели данных 24
2.5.2 Модель «сущность-связь» 24
2.5.3 Даталогические модели данных 25
2.5.4 Документальные модели данных 26
2.5.5 Фактографические модели данных 26
2.5.6 Модели данных на основе инвертированных файлов 26
2.5.7 Теоретико-графовые модели данных 27
2.5.8 Теоретико-множественные модели данных 28
2.5.9 Объектно-ориентированные модели данных 30
Контрольные вопросы 31
Список рекомендуемой литературы 31

ГЛАВА 3
Реляционные базы данных 32
3.1 Реляционная модель данных 32
3.1.1 Отношения 32
3.1.2 Ключи 33
3.1.3 Связывание отношений 34
3.1.4 Реляционные операции 35
3.2 Правила Кодда 38
3.3 Нормализация реляционных баз данных 40

3.3.1 Первая нормальная форма 41
3.3.2 Вторая нормальная форма 41
3.3.3 Третья нормальная форма 42
3.3.4 Четвертая нормальная форма 42
3.3.5 Пятая нормальная форма 43
3.3.6 Денормализация баз данных 43
Контрольные вопросы 44
Список рекомендуемой литературы 44

ГЛАВА 4
Повышение эффективности работы баз данных 45
4.1 Параллельные вычисления 45
4.1.1 Понятие транзакции 45
4.1.2 Уровни изоляции 47
4.1.3 Выполнение транзакций 48
4.1.4 Обработка транзакций 49
4.1.4.1 OLTP-системы 49
4.1.4.2 OLAP-системы 49
4.1.4.3 Мониторы транзакций 50
4.2 Оптимизация баз данных 50
4.3 Безопасность баз данных 53
4.3.1 Защита баз данных от несанкционированного доступа 55
4.3.2 Защита баз данных от несанкционированного использования ресурсов 56
4.3.3 Защита баз данных от некорректного использования ресурсов 57
4.3.4 Защита баз данных с помощью внесения избыточности 57
Контрольные вопросы 58
Список рекомендуемой литературы 58

ГЛАВА 5
Структурированный язык запросов SQL 59
5.1 Введение в SQL 59
5.2 Оператор SELECT 59
5.2.1 Выборка данных 61
5.2.2 Агрегатные функции 66
5.2.3 Группировка данных 67
5.2.4 Условный оператор 69
5.2.5 Соединение таблиц 70
5.2.6 Операции над множествами 73
5.3 Оператор INSERT 77

5.4 Оператор UPDATE 79
5.5 Оператор DELETE 80
5.6 Функции SQL 81
5.6.1 Математические функции 81
5.6.2 Строковые функции 83
5.6.3 Функции работы с датой и временем 84
Контрольные вопросы 87
Список рекомендуемой литературы 87

ГЛАВА 6
Разработка баз данных с помощью Microsoft Office Access 2007 88
6.1 Разработка таблиц в Microsoft Office Access 2007 88
6.2 Разработка форм в Microsoft Office Access 2007 91
6.3 Разработка отчетов в Microsoft Office Access 2007 93
6.4 Разработка запросов в Microsoft Office Access 2007 95
6.5 Использование диаграмм в Microsoft Office Access 2007 97
Контрольные вопросы 98
Список рекомендуемой литературы 98

ГЛАВА 7
Разработка баз данных с помощью технологии ADO.NET 99
7.1 Характеристика технологии ADO.NET 99
7.2 Создание базы данных для соединения с приложением 99
7.3 Проектирование интерфейса приложения 105
7.4 Программирование базы данных с помощью C# 106
Контрольные вопросы 115
Список рекомендуемой литературы 115

ГЛАВА 8
Разработка баз данных для глобальной сети Интернет 116
8.1 Разработка структуры базы данных MySQL 116
8.2 Разработка и тестирование запросов MySQL 119
8.3 Проектирование клиентского приложения с помощью PHP 120
Контрольные вопросы 129
Список рекомендуемой литературы 129

Заключение 130
Алфавитный указатель 131

Предисловие

Учебное пособие посвящено изучению одной из наиболее важных и перспективных современных информационных технологий – базы данных, и предназначено для студентов специальности «Информационно-измерительная техника и технологии», а также для инженеров, чья деятельность непосредственно связана с проектированием и сопровождением баз данных. Учебное пособие состоит из введения, восьми глав, заключения и алфавитного указателя. После каждой главы читателю предлагается ответить на ряд вопросов, связанных с изученным материалом, а также рекомендуется список литературы.

Глава 1 «Основы проектирования баз данных» посвящена рассмотрению общих вопросов, связанных терминологией систем управления баз данных.

Глава 2 «Концепции разработки баз данных» направлена на изучение структур хранения данных на физическом уровне и построения логических моделей данных.

Глава 3 «Реляционные базы данных» посвящена изучению реляционных баз данных, особое внимание в данной главе уделяется рассмотрению проблем нормализации реляционных баз данных.

Глава 4 «Повышение эффективности работы баз данных» направлена на изучение вопросов, связанных с методами и средствами оптимизации работы баз данных, реализацией параллельных вычислений, а также организацией безопасности информации.

В главе 5 «Структурированный язык запросов SQL» рассматриваются вопросы построения запросов к реляционным базам данных с помощью структурированного языка запросов SQL, а также примеры использования этих запросов.

В главе 6 «Разработка баз данных с помощью Microsoft Office Access 2007» описываются основные этапы проектирования баз данных в Microsoft Office Access 2007.

В главе 7 «Разработка баз данных с помощью технологии ADO.NET» раскрываются вопросы создания баз данных и клиентских приложение в Microsoft Visual Studio.

В главе 8 «Разработка баз данных для глобальной сети Интернет» рассматриваются вопросы создания баз данных MySQL и разработки клиентских приложений на языке программирования PHP.

Введение в базы данных

С давних времен люди пытались описать последовательность наблюдаемых ими явлений. Такое описание называют данными.

Издревле употреблялся вид хранения данных – на деревянных палочках с зарубками. В средние века бирки пользовали для учета и сбора налогов. Бирку разрезали на две продольные части, одна оставалась у крестьянина, другая – у сборщика налогов. По зарубкам на обеих частях и велся счет уплаты налога, который проверяли складыванием частей бирки. В Англии, например, этот способ записи налогов существовал до конца XVII в. При ликвидации старых налоговых обязательств крестьян на дворе лондонского казначейства был устроен костер из накопившихся бирок. Он оказался таким большим, что сгорело и само казначейство, а вместе с ним погиб и вделанный в стену образец английской меры длины, так что с тех пор англичане не знают точной длины своего фута.

Традиционно фиксация данных осуществляется с помощью конкретного средства общения, например, с помощью естественного языка на конкретном носителе. Обычно данные и их интерпретация фиксируются совместно, так как естественный язык достаточно гибок для представления того и другого. Примером может служить утверждение «Размер жесткого диска 160». Здесь «160» – данные, а «Размер жесткого диска» – их интерпретация.

Нередко данные и интерпретация разделены. Например, данные представлены в виде таблицы, шапка которой содержит интерпретацию, однако, такое разделение затрудняет работу с данными.

Применение компьютера для обработки данных обычно приводит к еще большему разделению данных и интерпретации, так как компьютер имеет дело с данными как таковыми. Большая часть интерпретирующей информации вообще не фиксируется в явной форме.

Основной причиной, по которой применение компьютера привело к отделению данных от интерпретации, является ограниченность обработки текстов на естественном языке – основном языке интерпретации данных. Это привело к тому, что память использовалась для хранения самих данных, а интерпретация традиционно возлагалась на приложение. Это существенно повышало роль программы, так как вне интерпретации данные не были пригодны для использования.

Жесткая зависимость между данными и использующими их приложениями создает серьезные проблемы в ведении данных и делает использования их менее гибкими.

В книге У. Девиса «Операционные системы» приведен пример: «Несколько лет назад почтовое ведомство (из лучших побуждений) пришло к решению, что все адреса должны обязательно включать почтовый индекс. Во многих вычислительных центрах это, казалось бы, незначительное изменение привело к ужасным последствиям. Добавление к адресу нового поля, содержащего шесть символов,

означало необходимость внесения изменений в каждую программу, использующую данные этой задачи в соответствии с изменившейся суммарной длиной полей. Тот факт, что какой-то программе для выполнения ее функций не требуется знания почтового индекса, во внимание не принимался: если в некоторой программе содержалось обращение к новой, более длинной записи, то в такую программу вносились изменения, обеспечивающие дополнительное место в памяти».

Активная деятельность по отысканию приемлемых способов обобществления непрерывно растущего объема информации привела к созданию в начале 60-х годов специальных программных комплексов, называемых «Системы управления базами данных» (СУБД).

История систем управления базами данных начинается с одного из самых значительных событий двадцатого века – полета на Луну. Североамериканская компания Rockwell заключила контракт с правительством США на участие в проекте Apollo. Построение космического корабля включает в себя сборку нескольких миллионов деталей, поэтому была создана система управления файлами, отслеживавшая информацию о каждой детали. Однако в ходе последующей проверки обнаружилась огромная избыточность. Выяснилось, что почти все данные повторяются в двух и более файлах.

Столкнувшись с задачей координации заказов на миллионы деталей, компания Rockwell в сотрудничестве с IBM в 1968 г. разработала автоматизированную систему заказов. Названная IMS (Information Management System – система управления информацией), она заложила основу концепции СУБД.

Ключевым новшеством IMS было разделение данных и функций деловой логики. Прикладные программисты получили возможность работать с информацией на логическом уровне, а база данных брала на себя задачу физического хранения. Подобное разделение труда привело к резкому скачку производительности.

Еще одним изобретением стал язык DL/I (Data Language / I). Это был специализированный язык составления нерегламентированных запросов к базе данных. Его появление сделало ненужным дорогостоящее программирование на таких языках, как COBOL и FORTRAN, популярных в то время.

Основной особенностью разработанных в это время СУБД являлось наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры. Файлы, снабженные описанием хранимых в них данных и находящиеся под управлением СУБД, стали называть «Базы данных» (БД).

На сегодняшний день базы данных – это одна из наиболее важных современных компьютерных технологий.

ГЛАВА 1
ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

В настоящей главе рассматриваются общие вопросы, связанные терминологией систем управления баз данных, описываются этапы проектирования баз данных.

1.1 Терминология в СУБД

Перед тем как приступить к изучению теоретических основ проектирования баз данных необходимо изучить терминологию. В общегосударственном стандарте ГОСТ 20886-85 «Организация данных в системах обработки данных» приводятся следующие определения основных понятий.

Организация данных – представление данных и управление данными в соответствии с определенными соглашениями.

Управление данными – совокупность функций обеспечения требуемого представления данных, их накопления и хранения, обновления, удаления, поиска по заданному критерию и выдачи данных.

Представление данных – характеристика, выражающая правила кодирования элементов и образования конструкций данных на конкретном уровне рассмотрения в вычислительной системе.

База данных (БД) – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

Блок данных – битовая последовательность, передаваемая как единое целое между устройствами вычислительной системы, системы телеобработки данных или вычислительной сети.

Язык манипулирования данными – язык, предназначенный для формулирования запросов на поиск, обмен данными между прикладной программой и базой данных, а также для расширения языка программирования либо как самостоятельный язык.

Система управления базами данных (СУБД) – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами.

В настоящее время СУБД должна обеспечивать возможность представления внутренней структуры данных, физическую и логическую независимость данных, минимальную избыточность данных, возможность быстрого поиска, эффективные языки запросов к данным, требования безопасности, надежности, конфиденциальности, целостности. Однако системы управления базами данных прошли несколь-

ко этапов, пока не стали удовлетворять всем вышеперечисленным современным требованиям.

1.2 Поколения СУБД

Принято выделять три поколения систем управления базами данных.

К первому поколению СУБД относятся сетевые и иерархические системы управления базами данных, широко распространенные в 70-е годы, получившие название системы управления базами данных первого поколения. Это были первые системы, предлагавшие развитую функциональность СУБД в рамках единой системы, с языками определения и манипулирования данными для набора записей.

Наиболее характерными особенностями этих систем являются следующие моменты. Все ранние системы не основывались на каких-либо абстрактных моделях. В таких системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД.

В 80-е годы на смену системам первого поколения пришло современное семейство реляционных СУБД, называемых системами управления базами данных второго поколения.

Реляционные СУБД сейчас являются наиболее популярными. К основным особенностям данного поколения СУБД относятся реляционная модель данных, универсальные нормализованные отношения, теоретически достаточные для представления данных любой предметной области. Однако в нетрадиционных приложениях в базе данных появляются сотни, если не тысячи таблиц, над которыми постоянно выполняются дорогостоящие операции соединения, необходимые для воссоздания сложных структур данных, присущих предметной области. Осознавая эти ограничения и недостатки реляционных систем, исследователи в области баз пришли к идее создания нового поколения СУБД.

Термин «системы следующего (или третьего) поколения» вошел в жизнь после опубликования группой известных специалистов в области БД «Манифеста систем баз данных третьего поколения». В целом можно сказать, что СУБД следующего поколения – это прямые наследники реляционных систем, имеющие ряд существенных расширений.

Например, системы управления базами данных третьего поколения ориентированы на расширенную реляционную модель, заключающуюся в использовании абстрактных типов данных, поддержке темпоральных запросов; отказе от требований нормализации.

Также к системам третьего поколения относятся системы дедуктивных баз данных, основанные на достижении расширяемости системы и ее адаптации к нуждам конкретных приложений путем использования стандартного механизма управления правилами.

Широкое внедрение Интернет в различные сферы деятельности ставит перед разработчиками СУБД новые проблемы, которые в будущем решат системы управления базами данных нового поколения. К таким проблемам можно отнести:

1) интеграция текста, данных, кода и потоков. В области СУБД основное внимание всегда уделялось организации, хранению, анализу и выборке структурированных данных. Развитие Интернет продемонстрировало важность более сложных типов данных: текстов, изображений, темпоральных, аудио- и видеоданных;

2) интеграция информации. В Интернет требуется производить интеграцию миллионов информационных источников. В связи с этим существует множество нерешенных проблем: семантическая неоднородность, неполнота и неточность данных, ограниченность доступа к конфиденциальным данным;

3) мультимедийные запросы. Очевидно, что объем мультимедийных данных (изображения, аудио, видео) значительно возрастает. Проблемой сообщества баз данных является создание простых способов анализа, обобщения, поиска и обозрения мультимедийной информации, относящейся к некоторому объекту.

1.3 Этапы проектирования баз данных

Рассмотрим процесс проектирования базы данных. Он состоит из трех основных этапов: концептуальное, логическое и физическое проектирование.

Первый этап процесса проектирования базы данных называется концептуальным проектированием базы данных. Он заключается в создании концептуальной модели данных для анализируемой предметной области. Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной СУБД, набор создаваемых прикладных программ, используемые языки программирования, а также от любых других особенностей физической реализации.

При разработке концептуальная модель данных постоянно подвергается мозговому штурму, тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных является источником информации для этапа логического проектирования базы данных.

Второй этап проектирования базы данных называется логическим проектированием базы данных. Его цель состоит в создании логической модели данных для исследуемой части предметной области. Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает лишь те конструкции данных и операции над ними, которые находятся в распоряжении СУБД.

На этом этапе уже должно быть известно, какая СУБД будет использоваться в качестве целевой – реляционная, сетевая, иерархическая или объектно-ориентированная. Однако на этом этапе игнорируются все остальные характеристики выбранной СУБД.

В процессе разработки логическая модель данных постоянно тестируется и проверяется на соответствие требованиям пользователей. Для проверки правильности логической модели данных используется метод нормализации. Нормализация гарантирует, что отношения, выведенные из существующей модели данных, не будут обладать избыточностью данных, способной вызвать нарушения в процессе обновления данных после их физической реализации.

Физическое проектирование является последним этапом создания проекта базы данных, при выполнении которого проектировщик принимает решения о способах реализации разрабатываемой базы данных. Физическое проектирование учитывает размещение и связь данных в среде хранения. Между логическим и физическим проектированием существует постоянная обратная связь, так как решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

Этапы концептуального и логического проектирования следует отделять от этапов физического проектирования, так как они несут абсолютно разные функции, выполняются в разное время, требуют совершенно разных навыков и опыта, поэтому для их реализации привлекаются специалисты различного профиля.

Проектирование базы данных – это итерационный процесс, который имеет свое начало, но не имеет конца и состоит из бесконечного ряда уточнений. Особо важную роль в общем процессе успешного создания системы играет концептуальное и логическое проектирование базы данных. Если на этих этапах не удастся получить полное представление о предметной области, то задача определения всех необходимых пользовательских представлений или обеспечения защиты базы данных становится чрезмерно сложной или даже неосуществимой.

Жизненный цикл разработки базы данных состоит из этапов, представленных на рисунке 1.1. Процесс разработки базы данных начинается с планирования разработки базы данных, заключающегося в поиске наиболее эффективного способа реализации этапов жизненного цикла системы. Затем необходимо определить требования к системе, найти диапазон действий и границ приложения базы данных, состав его пользователей и область применения. После этого необходимо собрать и проанализировать требования пользователей из всех возможных областей применения.

Полный цикл разработки включает концептуальное, логическое и физическое проектирование базы данных, которые описаны выше. После выбора наиболее подходящей СУБД для приложения базы данных, нужно определить пользовательский интерфейс и прикладные программы, которые будут использоваться в базе данных. Иногда необходимо создать рабочую модель приложения базы данных, которая позволит разработчикам или пользователям представить и оценить окончательный вид и способы функционирования системы.

Затем приложение базы данных тестируется с целью обнаружения ошибок, а также его проверки на соответствие всем требованиям, выдвинутым пользователями. После этого база данных эксплуатируется и в случае необходимости модифицируется.

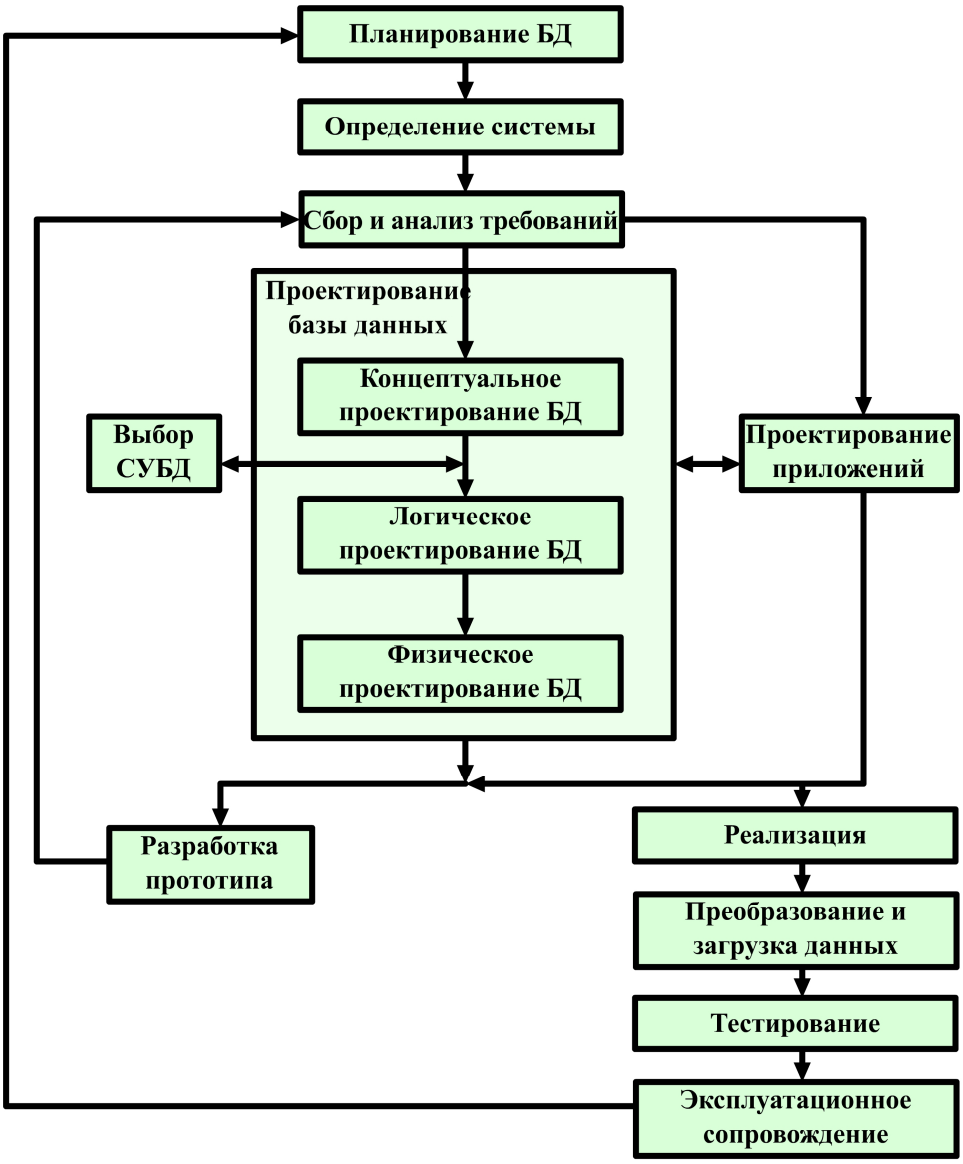


Рисунок 1.1 – Этапы жизненного цикла базы данных

Эти этапы не являются строго последовательными, а предусматривают в некоторых случаях возврат к предыдущим этапам с помощью обратных связей. Например, при проектировании базы данных могут возникнуть проблемы, для разрешения которых потребуется вернуться к этапу сбора и анализа требований или вообще планирования новой базы данных. Обратные связи могут возникать почти между всеми этапами.

Тезаурус

База данных, система управления базами данных, приложение базы данных, жизненный цикл базы данных.

Контрольные вопросы

1) Какое событие предопределило появление баз данных?
2) Сколько существует поколений БД?
3) Приведите пример данных и их интерпретации.
4) Дайте определение базы данных.
5) Какие действия выполняются на этапе концептуального проектирования баз данных?
6) Каково назначение языка манипулирования данными?
7) Назовите этапы жизненного цикла БД.
8) Что должна обеспечивать СУБД?
9) Какие требования выдвигаются к системам управления базами данных, публикуемых в глобальной сети Интернет?
10) Опишите этапы проектирования БД.

Список рекомендуемой литературы

1) Вербовецкий, А. А. Основы проектирования баз данных [Текст] / А. А. Вербовецкий. – М. : Радио и связь, 2000. – 542 с.
2) ГОСТ 20886-85 Организация данных в системах обработки данных [Текст] / – Введ. 1986–07–01. – М. : Изд-во стандартов, 1986.
3) Дейт, К. Дж. Введение в системы баз данных [Текст] / К. Дж. Дейт. – М. : Вильямс, 2005. – 1328 с.
4) Илюшечкин, В. М. Основы использования и проектирования баз данных [Текст] / В. М. Илюшечкин. – М. : Высшее образование, 2009. – 213 с.
5) Кузин, А. В. Базы данных [Текст] / А. В Кузин, С. В. Левонисова. – М. : Академия, 2008. – 320 с.
6) Кузнецов, С. Д. Основы баз данных [Текст] : учебное пособие / С. Д. Кузнецов. – М. : Бином, 2007. – 484 с.
7) Лебедева, Т. Ф. Базы данных [Текст] : учебное пособие / Т. Ф. Лебедева ; Рос. гос. торг.-эк. ун-т. – Кемерово : Изд-во РГТЭУ, 2006. – 99 с.

ГЛАВА 2
КОНЦЕПЦИИ РАЗРАБОТКИ БАЗ ДАННЫХ

В настоящей главе рассматриваются вопросы, связанные со структурой хранения данных на физическом уровне, а также построения логических моделей данных.

2.1 Модель ANSI/SPARC

Первая попытка создания стандартной терминологии и общей архитектуры СУБД была предпринята в 1971 году. Группа DBTG признала необходимость использования двухуровневого подхода, построенного на основе системного представления и пользовательских представлений. Сходные терминология и архитектура были предложены в 1975 году Комитетом планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США (American National Standard Institute). Комитет ANSI/SPARC признал необходимость использования трехуровневого подхода к созданию баз данных. Основное внимание в них было сконцентрировано на необходимости воплощения независимого уровня для изоляции программ от особенностей представления данных на более низком уровне. Хотя модель ANSI/SPARC не стала стандартом, она все еще представляет собой основу для понимания некоторых функциональных особенностей СУБД.

Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления.

При таком разделении выдвигается ряд требований к СУБД, которые делают ее работу более эффективной. Например, в модели ANSI/SPARC каждый пользователь должен иметь возможность обращаться к одним и тем же данным, реализуя свое собственное представление о них. Каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей. Взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных. Администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.

Схема трехуровневой архитектуры ANSI/SPARC показана на рисунке 2.1. Уровень, на котором данные воспринимаются пользователями, называется внешним уровнем, тогда как СУБД и операционная система воспринимают данные на внутреннем уровне. Концептуальный уровень представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости друг от друга. Более подробно рассмотрим физическую организацию данных и логические модели данных, составляющих основу концептуального и физического уровня модели ANSI/SPARC.

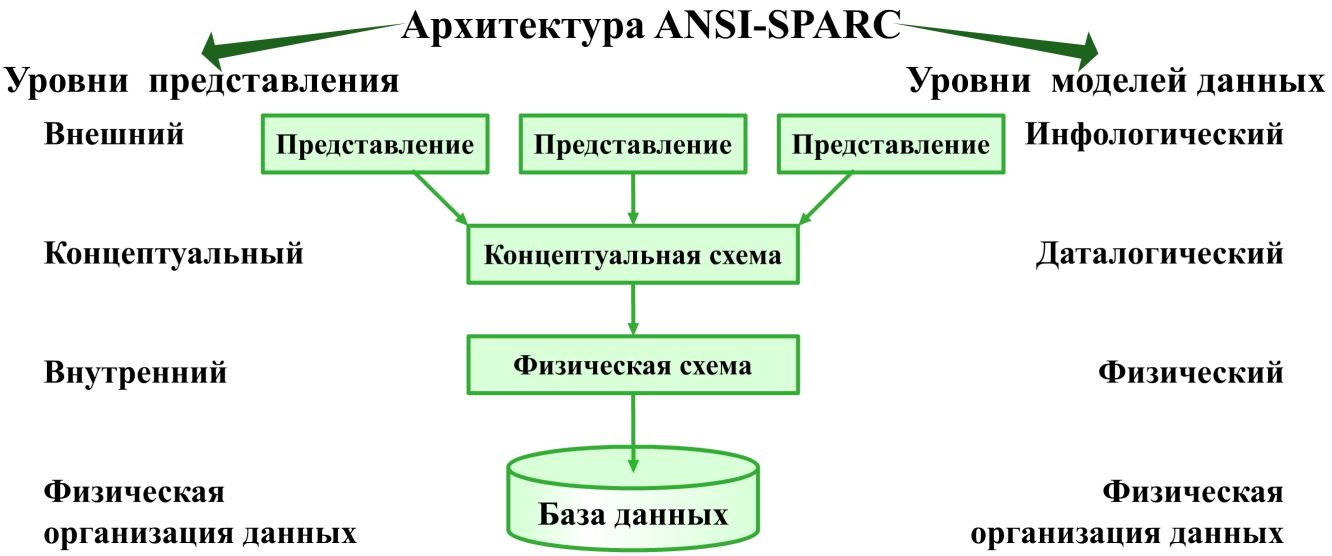


Рисунок 2.1 – Трехуровневая архитектура ANSI/SPARC

2.2 Физическая организация данных

Физическая организация данных представляет собой организацию данных, учитывающую размещение и связь данных в среде хранения.

Среди самых важных характеристик любой базы данных следует назвать производительность, надежность и простоту администрирования. Знание того, как большинство СУБД физически хранят данные во внешней памяти, представления о параметрах этого хранения и соответствующих методов доступа может очень помочь при проектировании баз данных, обладающих заданной производительностью.

Выделяют несколько проблем физического представления данных, которые необходимо преодолеть, чтобы работа базы данных была максимально эффективной.

Во-первых, необходимо решить, как осуществлять поиск нужной записи. Для решения этой проблемы нужно установить соответствие между логической записью и адресом физической записи. Во-вторых, необходимо задаться вопросом, как организовать данные, чтобы их поиск был эффективным. В-третьих, как добавлять новые записи к данным, уничтожить старые записи и при этом не нарушить системы адресации и поиска.

К основным факторам, влияющим на физическую организацию данных, принято относить показатели, которые показаны на рисунке 2.2.

Исторически первыми системами хранения и доступа были файловые структуры и системы управления файлами, которые фактически являлись частью операционных систем. СУБД создавала над этими файлами свою надстройку, которая

позволяла организовать всю совокупность файлов таким образом, чтобы она работала как единое целое и получала централизованное управление от СУБД. При этом непосредственный доступ осуществлялся на уровне файловых команд, которые СУБД использовала при манипулировании файлами.

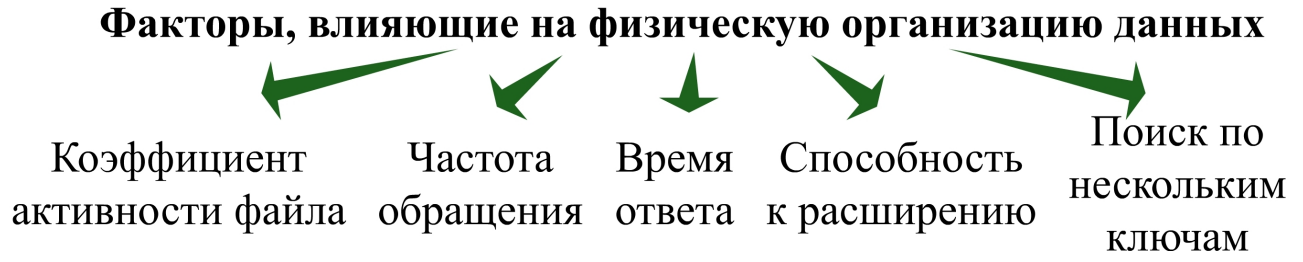


Рисунок 2.2 – Факторы, влияющие на физическую организацию данных

Однако механизмы буферизации и управления файловыми структурами не приспособлены для решения задач собственно СУБД, так как создавались для традиционной обработки файлов, и с ростом объемов хранимых данных они стали неэффективными для использования СУБД. При этом постепенно произошел переход от базовых файловых структур к непосредственному управлению размещением данных на внешних носителях самой СУБД. В таком случае механизмы, применяемые в файловых системах, перешли во многом и в новые системы организации данных во внешней памяти.

Структурой хранения данных называется любое упорядоченное расположение данных на диске. На рисунке 2.3 приведена классификация структур хранения данных. **Методом доступа** называют совокупность соглашений и средств, с помощью которых реализуется заданный вид доступа к физическим записям набора данных.

Рисунок 2.3 – Классификация структур хранения информации

На некоторых устройствах технически могут быть организованы файлы только последовательного доступа. Файлы с переменной длиной записи также всегда являются файлами последовательного доступа и могут быть организованы двумя способами: конец записи отмечается специальным маркером, в начале каждой записи записывается ее длина (рисунок 2.4).

Рисунок 2.4 – Последовательный доступ к данным

Файлы с постоянной длиной записи, расположенные на устройствах прямого доступа, являются файлами прямого доступа. В этих файлах физический адрес расположения нужной записи может быть вычислен по номеру записи. Файлы прямого доступа обеспечивают наиболее быстрый доступ к произвольным записям, и их использование является наиболее перспективным в системах управления базами данных (рисунок 2.5).

Рисунок 2.5 – Прямой доступ к данным

Однако чаще всего в базе данных необходим поиск по ключам, а не по номеру записи, и номер записи, необходимый для прямого доступа, в этом случае неизвестен. При организации файлов прямого доступа в некоторых случаях возможно построение функции, которая по значению ключа однозначно вычисляет номер записи (рисунок 2.6).

Рисунок 2.6 – Функциональный доступ к данным

Нужно отметить, что часто не удается построить взаимно-однозначное соответствие между значениями ключа и номерами записей, поэтому применяют различные методы хеширования и создают специальные хэш-функции. Общей идеей методов хэширования является применение к значению ключа некоторой функции

свертки (хэш-функции), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи.

В самом простом, классическом случае, свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значения свертки. При возникновении коллизий образуются цепочки переполнения. Новая запись заносится в область переполнения на первое свободное место, а в записи-синониме (с тем же значением хэш-функции), которая находится в основной области, делается ссылка на адрес вновь размещенной записи в области переполнения. Главным ограничением этого метода является фиксированный размер таблицы. Если таблица заполнена слишком сильно или переполнена, то возникнет много цепочек переполнения, и главное преимущество хеширования – доступ к записи почти всегда за одно обращение к таблице – будет утрачено. Расширение таблицы требует ее полной переделки на основе новой хэш-функции (со значением свертки большего размера).

В случае базы данных такие действия являются абсолютно неприемлемыми. Поэтому обычно вводят промежуточные таблицы-справочники, содержащие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что значительно проще.

Несмотря на высокую эффективность хэш-адресации, в файловых структурах не всегда удается найти соответствующую функцию, поэтому при организации доступа по первичному ключу широко используются индексные файлы. Ключевым понятием при построении индексных файлов является индекс доступа. **Индекс доступа** – это совокупность данных, обеспечивающих соответствие между значениями ключей порций данных и адресами этих порций или областей пространства памяти, в которых они находятся, с целью повышения скорости доступа к порции данных.

Индексные файлы можно представить как таблицы указателей к основному файлу с записями, причем для одного основного файла можно построить несколько индексных файлов по различным ключам.

Различают два типа индексных файлов: с плотным индексом (индексно-прямые) и с разреженным индексом (индексно-последовательные).

Плотным индексом называется индекс доступа, в котором, как в таблице, хранятся сведения о местоположении порции данных для каждого допустимого значения ключа поиска.

Разреженным индексом называется индекс доступа, в котором хранятся сведения о местоположении порции данных лишь для некоторого подмножества значений ключа поиска, а сведения о местоположении порций данных с остальными значениями устанавливают косвенным образом с учетом определенной упорядоченности этих порций.

Наиболее популярным подходом к организации индексов в базах данных является использование техники B-деревьев. С точки зрения внешнего логического

представления B-дерево – это сбалансированное дерево во внешней памяти. Сбалансированность означает, что длина пути от корня дерева к любому его листу одна и та же. Построение B-деревьев связано с идеей построения индекса над уже построенным индексом. Если построить файл с неплотным индексом, то, рассматривая его как основной файл, над которым нужно снова построить файл с неплотным индексом, а потом снова над новым индексом строим следующий и так до того момента, пока не останется всего один индексный блок.

Если индексные файлы используются для ускорения доступа по первичному ключу, то для ускорения доступа по вторичному ключу используются структуры, называемые инвертированными списками. Вторичным ключом является атрибут или набор атрибутов, которому соответствует несколько искомых записей.

Инвертированный список в общем случае – это трехуровневая индексная структура. На первом уровне находится файл или часть файла, в которой упорядоченно расположены значения вторичных ключей. Каждая запись с вторичным ключом имеет ссылку на номер первого блока в цепочке блоков, содержащих номера записей с данным значением вторичного ключа. На втором уровне находится цепочка блоков, содержащих номера записей с одним и тем же значением вторичного ключа. При этом блоки второго уровня упорядочены по значениям вторичного ключа. На третьем уровне находится основной файл с записями. Механизм доступа к записям по вторичному ключу можно представить следующим образом.

Во-первых, в области первого уровня ищется заданное значение вторичного ключа. Во-вторых, по ссылке считываются блоки второго уровня, содержащие номера записей с заданным значением вторичного ключа. В-третьих, в рабочую область пользователя прямым доступом загружается содержимое всех записей с заданным значением вторичного ключа.

Для одного основного файла может быть создано несколько инвертированных списков по разным вторичным ключам. Однако при модификации основного файла требуется внести изменения во все инвертированные списки. Поэтому можно утверждать, что построение инвертированных списков ускоряет процесс доступа только в том случае, если база данных стабильна и ее содержимое не изменяется.

2.3 Обмен информацией в базе данных

Реляционные СУБД хранят разновидности объектов во внешней памяти, показанные на рисунке 2.7. Хранение данных во внешней памяти в известных СУБД организовано очень похожим образом. Основными единицами физического хранения являются блоки данных.

Рисунок 2.7 – Разновидности объектов внешней памяти

Блок данных – это битовая последовательность, передаваемая как единое целое между устройствами вычислительной системы, системы телеобработки данных или вычислительной сети.

Размер блока оказывает большое влияние на производительность базы данных – при больших размерах скорость операций чтения/записи растет, однако возрастают накладные расходы на хранение, и снижается эффективность индексных просмотров. Меньший размер блока позволяет более экономно расходовать память, но вместе с тем относительно дорог. Длинные блоки лучше использовать для больших объектов данных: полнотекстовые фрагменты, мультимедиа-объекты, изображения, длинные строки. Короткие блоки лучше подходят для значений числовых типов, недлинных строк, значений даты и времени. Следует также учитывать размер блока операционной системы, он должен быть кратен размеру блока базы данных.

Основной единицей осуществления обмена данных является **страница данных**. Все данные хранятся постранично. При табличном хранении данные на одной странице являются однородными, т. е. страница может содержать только данные или только индексы. Все страницы данных имеют одинаковую структуру, представленную на рисунке 2.8.

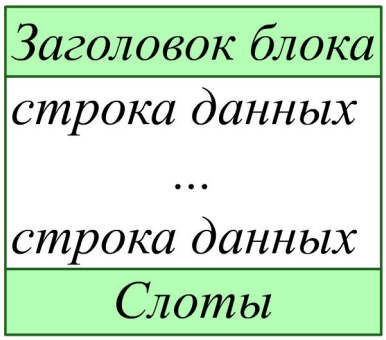


Рисунок 2.8 – Структура страницы данных

Заголовок страницы содержит следующую информацию: логический номер страницы; логические номера следующей и предыдущей страниц в цепочке; идентификатор объекта данных, которому принадлежит страница; номер следующей свободной строки на странице; тип страницы.

Слоты характеризуют размещение строк данных на странице. Слот – это 4-байтовое слово, 2 байта соответствуют смещению строки данных на странице и 2 байта – длина строки. В базе данных каждая строка имеет уникальный идентификатор в рамках всей базы данных – номер строки, он имеет размер 4 байта и состоит из номера страницы и номера строки на странице. При упорядочении строк на страницах не происходит физического перемещения строк, все манипуляции происходят со слотами. При переполнении страниц создается специальный вид страниц, называемых страницами остатка. Строки, не уместившиеся на основной странице, связываются со своим продолжением на страницах остатка с помощью ссылок-указателей, которые содержат номер страницы и номер слота на странице.

2.4 Этапы доступа к базе данных

Последовательность действий при доступе к данным можно описать следующим образом:

1) сначала в СУБД определяется искомая запись, а затем для ее извлечения запрашивается диспетчер файлов;

2) диспетчер файлов одним из рассмотренных способов адресации определяет страницу, на которой находится искомая запись, а затем для ее извлечения запрашивается диспетчер дисков;

3) диспетчер дисков определяет физическое положение искомой страницы на диске и посылает запрос на ввод/вывод данных.

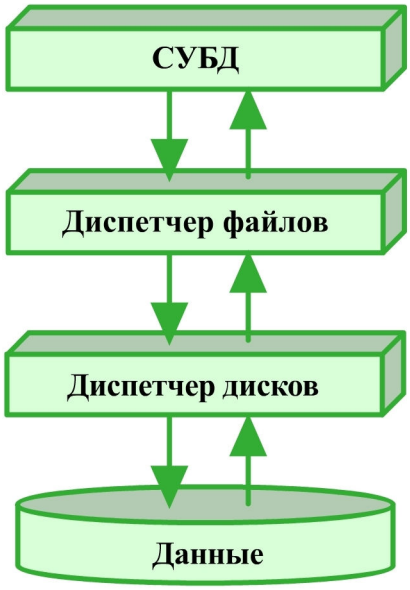


Рисунок 2.9 – Схема доступа к данным

База данных при таком подходе имеет определенный дуализм. С точки зрения СУБД база данных выглядит как набор записей, которые можно просматривать с помощью диспетчера файлов. С точки зрения диспетчера файлов база данных выглядит как набор страниц, которые можно просматривать с помощью диспетчера.

Диспетчер дисков часто бывает компонентом операционной системы, с помощью которого выполняются все операции ввода/вывода, используя физические адреса записей. Однако диспетчер файлов не обязательно должен знать физические адреса записей, достаточно рассматривать диск как набор страниц фиксированного размера с уникальным идентификатором набора страниц. Соответствие физических адресов на диске и номера станиц достигается с помощью применения диспетчера дисков.

2.5 Модели данных

После рассмотрения аспектов физического хранения данных, необходимо изучить возможности работы с данными на логическом уровне, которые обеспечивают модели данных. Модель данных является основополагающим понятием концепции разработки баз данных.

Модель данных – это совокупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения.

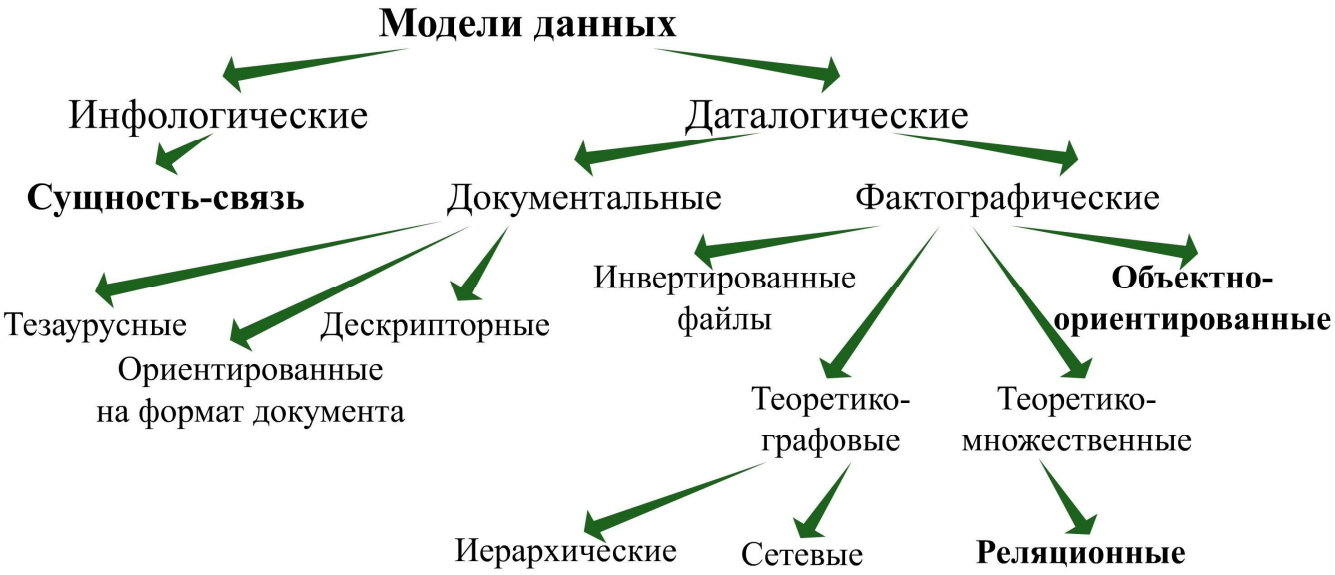


Рисунок 2.10 – Классификация моделей данных

Модели данных с течением времени претерпевали значительные изменения в соответствии с требованиями, выдвигаемыми вновь разрабатываемым информационным технологиям. На смену дескрипторным файловым моделям приходили сетевые и иерархические, затем и эти модели данных были потеснены реляционными моделями, которые в свою очередь сегодня уступают место объектно-ориентированным, объектно-реляционным, многомерным моделям данных. Классификация моделей данных приведена на рисунке 2.10, рассмотрим более подробно представленные модели данных.

2.5.1 Инфологические модели данных

Инфологическая модель данных отображает предметную область в концепции, понятной человеку, такая модель не зависит от параметров среды хранения данных. Инфологическая модель направлена на хранение совокупности информационных объектов и их структурных связей. Существует множество подходов к построению таких моделей. Наиболее популярной из них является модель «сущность-связь» (ER).

2.5.2 Модель «сущность-связь»

Модель «сущность-связь» была введена Питером Ченом в 1976 г., который предложил представлять предметную область с помощью графических диаграмм, включающих небольшое число разнородных компонентов. Основными понятиями модели «сущность-связь» являются сущность, связь и атрибут.

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступной. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности представляет собой имя типа. При определении типа сущности необходимо гарантировать, что каждый экземпляр сущности может быть отличим от любого другого экземпляра той же сущности.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя типами сущностей. В любой связи в соответствии с существующей парой связываемых сущностей выделяются два конца. На каждом конце связи указываются имя конца связи, степень конца связи (сколько экземпляров данного типа сущности должно присутствовать в каждом экземпляре данного типа связи), обязательность связи (должен ли любой экземпляр данного типа сущности участвовать в некотором экземпляре данного типа связи).

Связь представляется в виде ненаправленной линии, соединяющей две сущности или ведущей от сущности к ней же самой. При этом в месте соединения связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много экземпляров сущности,

одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный – прерывистой линией.

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем.

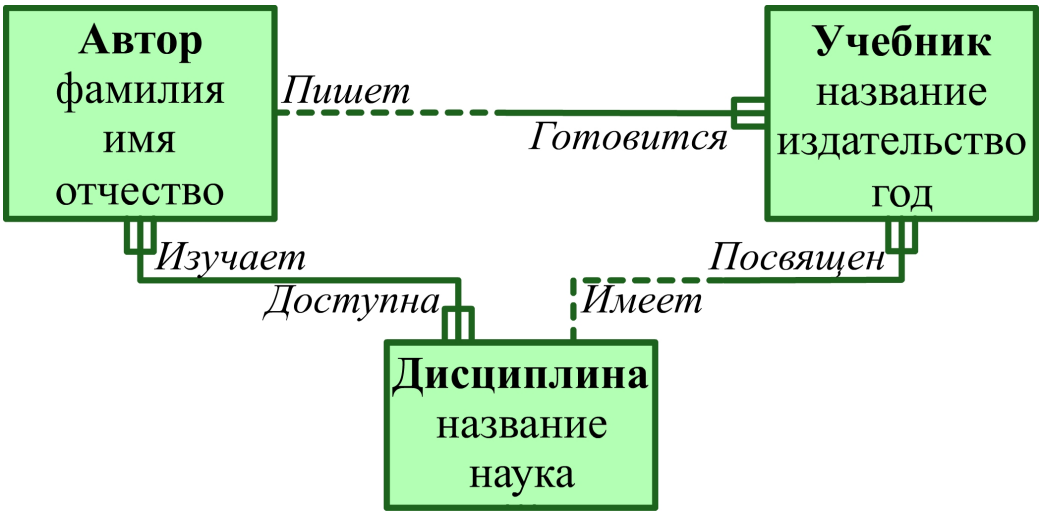


Рисунок 2.11 – Пример модели «сущность-связь»

На рисунке 2.11 представлена диаграмма модели «сущность-связь», отражающая связи между авторами, учебниками и дисциплинами при условии, что авторство учебника принадлежит одному человеку. Трактовка данной диаграммы состоит в следующем. Каждый автор изучает дисциплины, причем все дисциплины доступны для изучения авторам. Каждый автор может написать один или много учебников, каждый учебник готовится одним автором. Каждая дисциплина может иметь один или много учебников, каждый учебник посвящен одной дисциплине.

Простота и наглядность представления концептуальных схем баз данных в модели «сущность-связь» привели к ее широкому распространению и популярности среди разработчиков баз данных.

2.5.3 Даталогические модели данных

Даталогические модели данных получили наибольшее распространение на сегодняшний день. Среди таких моделей выделяют документальные и фактографические модели данных. Рассмотрим их более подробно.

2.5.4 Документальные модели данных

Документальные модели данных соответствуют представлению о слабо структурированной информации, ориентированной на свободные форматы текста на естественном языке.

Модели, ориентированные на формат документа, связаны со стандартным языком разметки XML. Такая модель данных популярна среди разработчиков приложений, которые используют стандартную выгрузку данных для дальнейшего их использования другими программами. Жесткая структура документа и универсальность подхода одновременно являются преимуществом и недостатком моделей данных, ориентированных на формат документа.

Дескрипторные модели являются самыми простыми из документальных моделей и широко применялись на ранних стадиях использования документальных баз данных. В этих моделях каждому документу соответствует дескриптор, имеющий жесткую структуру. Дескриптор описывал документ в соответствии с теми характеристиками, которые требуются для работы с документами в разрабатываемой базе данных.

Тезаурусные модели основаны на принципе организации словарей и содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели активно используются в многоязыковых переводчиках.

2.5.5 Фактографические модели данных

Фактографические модели данных в отличие от документальных моделей соответствуют представлению о структурированной информации. К таким моделям относятся модели данных на основе инвертированных списков, теоретико-графовые и теоретико-множественные модели данных, а также объектно-ориентированные модели данных. Рассмотрим их более подробно.

2.5.6 Модели данных на основе инвертированных файлов

В шестидесятые годы двадцатого века появились системы управления базами данных на основе инвертированных файлов, отличающиеся простотой организации и наличием языков манипулирования данными.

Простейшая база данных, основанная на применении инвертированных файлов, представляет собой набор файлов, доступ к которым осуществляется с помощью программ, в которых четко прописано, в каких позициях в файле находится необходимая информация. Недостатком такого подхода является то, что он не удовлетворяет модели ANSI/SPARC, так как и логическая, и физическая структура хранения данных скрыта в программе. По мере расширения системы растет коли-

чество обрабатываемых файлов и программ, что приводит к дублированию данных. При этом безопасность системы обеспечивается лишь на уровне операционной системы, что приводит к снижению эффективности работы.

2.5.7 Теоретико-графовые модели данных

Теоретико-графовые модели данных являются ранними моделями данных, которые доминировали в области проектирования баз данных, начиная с семидесятых годов двадцатого века.

Первая версия иерархической модели данных появилась в 1968 г. Простота организации, наличие заранее заданных связей между сущностями, сходство с физическими моделями данных позволяли добиваться приемлемой производительности иерархических систем управления базами данных на медленных компьютерах с малым объемом памяти.

Иерархические базы данных поддерживают древовидную организацию информации. Связи между записями выражаются в виде отношений «предок-потомок», а у каждой записи есть ровно одна родительская запись. Это помогает поддерживать ссылочную целостность. Когда запись удаляется из дерева, все ее потомки также должны быть удалены. Но, если данные не имеют древовидной структуры, то возникают сложности при построении базы данных.

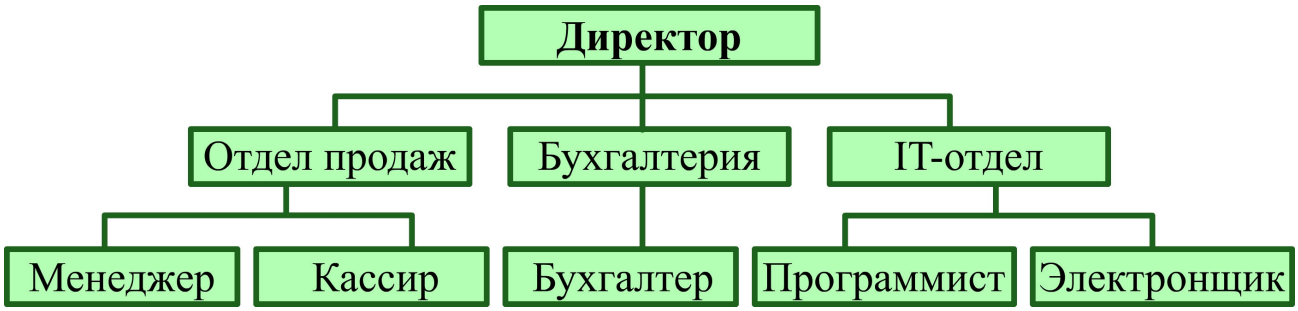


Рисунок 2.12 – Пример иерархической модели данных

Сетевые модели также создавались для мало ресурсных компьютеров. Сетевая модель расширяет иерархическую модель, позволяя группировать связи между записями в множестве. С логической точки зрения связь – это не сама запись. Связи лишь выражают отношения между записями. Как и в иерархической модели, связи ведут от родительской записи к дочерней, но на этот раз поддерживается множественное наследование. Применение сетевой модели данных позволило увеличить производительность систем управления базами данных, но существенно их усложнила.

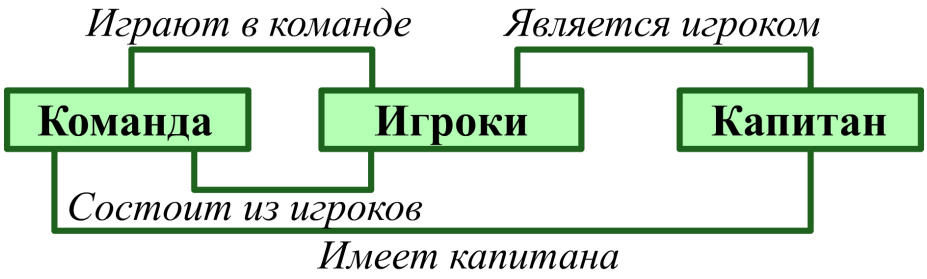


Рисунок 2.13 – Пример сетевой модели данных

Сложность практического использования иерархических и сетевых систем управления базами данных заставляла искать иные способы представления данных.

2.5.8 Теоретико-множественные модели данных

В июне 1970 г. сотрудник IBM Эдгар Кодд опубликовал документ «Реляционная модель для больших банков совместно используемых данных», который перевернул теорию баз данных и принес доктору Кодду награду Тьюринга в 1981 году. В реляционных моделях данных ключевым понятием является отношение (от английского слова «relation»), представляющее собой множество элементов, называемыми кортежами, каждому из которых соответствует атрибут. Детально реляционная модель данных будет рассмотрена далее. Следует отметить, что достоинством реляционной модели данных является простота, понятность и удобство физической реализации.

Основным недостатком реляционной модели данных является сложность описания иерархических и сетевых связей, однако это не помешало реляционной модели широко использоваться разработчиками баз данных.

В настоящее время кроме реляционной модели в чистом виде СУБД поддерживают ее расширения, представляющие собой смешанные и дополненные модели, к таким относятся постреляционные и многомерные модели.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в кортежах отношений с помощью использования многозначных атрибутов, значения которых состоят из подзначений. Набор значений многозначных атрибутов считается самостоятельным отношением, встроенным в основное отношение. Кроме обеспечения вложенности атрибутов постреляционная модель поддерживает ассоциированные многозначные атрибуты.

Недостатком постреляционной модели данных является сложность решения проблемы обеспечения целостности данных.

Следующим этапом развития теоретико-множественных моделей данных являются многомерные модели, которые стали активно использоваться в девяностые годы двадцатого века для решения задач анализа и принятия решений. Основными понятиями, используемыми в многомерных системах управления базами данных, являются агрегируемость, историчность и прогнозируемость данных, а также измерение и ячейка. Рассмотрим их более подробно.

Агрегируемость данных – это рассмотрение информации на различных уровнях ее обобщения.

Историчность данных предполагает обеспечение высокого уровня статичности данных и их взаимосвязей, а также привязки данных ко времени.

Прогнозируемость данных подразумевает задание функций прогнозирования, и их применение к различным интервалам времени.

Многомерность модели данных выражается в многомерном логическом представлении структуры данных, что приводит к более высокой наглядности и информативности.

Измерение представляет собой множество однотипных данных, образующих одну из граней гиперкуба. Ячейкой называют поле, значение которого однозначно определяется фиксированным набором измерений. Приведем пример базы данных, использующей многомерную модель данных.

Рисунок 2.14 – Пример многомерной модели данных

На рисунке 2.14 приведен пример многомерной модели данных. Каждое значение ячейки «Температура» однозначно определятся географическим месторасположением, днем и временем суток. Для решения практических задач необходимо строить гиперкубы с большим количеством измерений.

2.5.9 Объектно-ориентированные модели данных

Объектно-ориентированные модели данных возникли сравнительно давно. Публикации появлялись уже в середине восьмидесятых годов прошлого века. Однако наиболее бурно данное направление развивается в последние годы. В настоящее время ведется много экспериментальных и производственных работ в области объектно-ориентированных СУБД.

В объектно-ориентированной модели данных между записями базы данных и функциями их обработки устанавливаются взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках программирования.

Основными понятиями, используемыми в объектно-ориентированных моделях данных, являются полиморфизм, наследование и инкапсуляция. Рассмотрим их более подробно.

Инкапсуляция ограничивает область видимости имени свойства пределами того объекта, в котором оно определено. Смысл такого свойства будет определяться тем объектом, в который оно инкапсулировано.

Наследование распространяет область видимости свойства на всех потомков объекта.

Полиморфизм означает возможность иметь в объектах разных типов методы с одинаковыми именами.

Данные в таких объектно-ориентированных системах управления базами данных способны принять вид любой структуры, которую можно описать на используемом языке программирования.

Отношения между объектами также могут быть сколь угодно сложными. При этом решается несколько очень важных проблем. Во-первых, сложные информационные структуры выражаются в них лучше, чем в реляционных базах данных, а во-вторых, устраняется необходимость транслировать данные из того формата, который поддерживается в СУБД.

Недостатком же объектно-ориентированных баз данных является их тесная связь с применяемым языком программирования, низкая скорость выполнения запросов и сложность работы с данными.

Тезаурус

Трехуровневая архитектура, структура хранения информации, страница, строка, слот, индекс, хэш-функция, модель данных, сущность, связь, атрибут, реляционная модель данных, многомерность данных, измерение данных, инкапсуляция данных, наследование данных, полиморфизм данных.

Контрольные вопросы

1) Какова цель создания трехуровневой архитектуры ANSI/SPARC?
2) Назовите основные проблемы физического представления данных?
3) Как организуется последовательный доступ к данным?
4) Дайте определение разряженного индекса доступа.
5) Раскройте суть использования техники построения B-деревьев?
6) Каково назначение слотов в структуре страницы данных?
7) Опишите последовательность действий при доступе к базе данных.
8) Какие недостатки присущи теоретико-графовым моделям данных?
9) Приведите пример многомерной базы данных.
10) В чем смысл инкапсуляции по отношению к объектно-ориентированной модели данных?

Список рекомендуемой литературы

1) Карпова, Т. С. Базы данных: модели, разработка, реализация [Текст] / Т. С. Карпова. – СПб. : Питер, 2002. – 304 с.
2) Кузнецов, С. Д. Основы баз данных [Текст] : учебное пособие / С. Д. Кузнецов. – М. : Бином, 2007. – 484 с.
3) Кузнецов, С. Д. Базы данных. Модели и языки [Текст] / С. Д. Кузнецов. – М. : Бином, 2008. – 720 с.
4) Лебедева, Т. Ф. Базы данных [Текст] : учебное пособие / Т. Ф. Лебедева ; Рос. гос. торг.-эк. ун-т. – Кемерово : Изд-во РГТЭУ, 2006. – 99 с.
5) Малыхина, М. П. Базы данных: основы, проектирование, использование [Текст] / М. П. Малыхина. – СПб. : BHV, 2007. – 528 с.
6) Фрост, Р. Проектирование и разработка баз данных. Визуальный подход / Р. Фрост, Д. Дей, К. Ван Слайк. – М. : Пресс, 2007. – 592 с.
7) Хомоненко, А. Д. Базы данных [Текст]: учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. – СПб. : Корона принт, 2004. – 736 с.

ГЛАВА 3
РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ

В настоящей главе рассматриваются вопросы проектирования и эксплуатации реляционных баз данных, особое внимание уделяется рассмотрению проблем нормализации реляционных баз данных.

3.1 Реляционная модель данных

Как было отмечено выше, реляционная модель данных была предложена сотрудником фирмы IBM Эдгаром Коддом, изложившим основные идеи в статье «Реляционная модель для больших банков совместно используемых данных». Реляционная модель данных позволила снять недостатки сетевых и иерархических моделей, а наглядность и строгая теоретическая проработка принесли большую популярность модели среди разработчиков баз данных.

3.1.1 Отношения

Опишем основные понятия, относящиеся к реляционной модели. Реляционная модель основана на математическом понятии **отношения**, физическим представлением которого является таблица.

Имеются различные интерпретации реляционных терминов, основные из которых представлены на рисунке 3.1. В зависимости от контекста будем пользовать один из этих подходов.

Реляционные термины	Табличный вариант	Файловый вариант	Объектная модель
отношение	таблица	файл	класс
кортеж	строка	запись	объект
атрибут	столбец	поле	свойство

Рисунок 3.1. – Альтернативные варианты терминов

Сущность описывает объект, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении.

Атрибут представляет собой свойство, описывающее сущность. В структуре отношения каждый атрибут имеет название и тип.

Кортеж определяется как отдельная сущность.

Доменом является набор всех возможных значений определенного атрибута отношения.

Схема отношения представляет собой список имен атрибутов.

Реляционная модель предполагает организацию данных только в виде таблиц. Таблица состоит из строк и столбцов (рисунок 3.2).

Рисунок 3.2 – Пример табличной организации данных

В реляционной модели данных к отношениям предъявляются определенные требования, выраженные в принципе информационной неделимости: каждое значение отношения реляционной модели может содержать только одну порцию данных; и принципе информационного кодирования: недопустимо, чтобы в значении отношения реляционной модели содержалось более одной порции данных.

3.1.2 Ключи

Связи между кортежами реализуются в виде ключей. **Ключ** – это определенным образом помеченный атрибут отношения. Ключ может быть составным, то есть состоять из нескольких атрибутов отношения. Ключи обычно используют для исключения дубликатов, упорядочения кортежей, ускорения работы с кортежами отношения, организации связывания отношений.

Для того чтобы его значения были связаны с атрибутом другого отношения, необходимо установить между атрибутами виртуальную связь с помощью первичных и внешних ключей.

Первичный ключ – это ключ, значения которого однозначно идентифицируют каждый из его кортежей в заданном отношении.

Внешний ключ – это ключ, который соответствует первичному ключу другого отношения.

Когда атрибут или набор атрибутов помечается как ключ, дополнительно создается индекс. **Индекс** хранит список значений ключа и указатели на кортежи,

содержащие эти значения. Наличие индекса позволяет системе управления базами данных быстро находить нужные кортежи. В целом индексы ускоряют выполнение операций чтения, но замедляют выполнение операций записи, так как при этом необходимо обновлять индекс.

3.1.3 Связывание отношений

Связывание отношений определяет их подчиненность. Существуют три вида связывания отношений: «один – к одному», «один – ко многим» и «много – ко многим». Рассмотрим более подробно каждый из этих видов.

Связывание отношений вида **«один – к одному»** обычно используется при разбиении отношений с большим числом атрибутов на несколько частей. В этом случае в одном отношении остаются атрибуты с наиболее важной и часто используемой информацией, а остальные атрибуты переносятся в другие отношения. Пример такого связывания приведен на рисунке 3.3, в котором одному кортежу одного отношения соответствует один кортеж другого отношения.

Рисунок 3.3 – Пример связывания отношений вида «один – к одному»

Связывание отношений вида **«один – ко многим»** означает, что одному кортежу одного отношения может соответствовать несколько кортежей другого отношения, в частном случае ни одной. Такой вид связывания встречается наиболее часто. Пример связывания отношений вида «один – ко многим» приведен на рисунке 3.4.

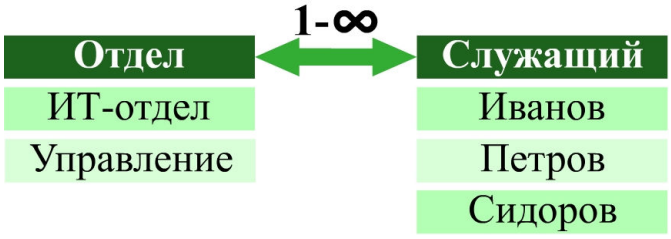


Рисунок 3.4 – Пример связывания отношений вида «один – ко многим»

Связывание отношений вида **«много – ко многим»** используется тогда, когда одному кортежу одного отношения соответствует несколько кортежей другого отношения и наоборот. На практике связывание отношений вида «много – ко мно-

гим» используется достаточно редко и для реализации такого отношения в реляционной модели преобразуется в связывание нескольких отношений вида «один – ко многим». Пример такого связывания приведен на рисунке 3.5.

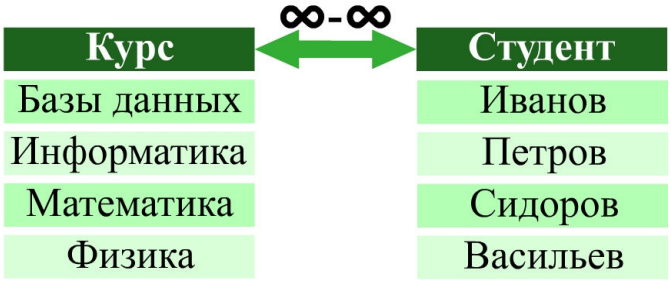


Рисунок 3.5 – Пример связывания отношений вида «много – ко многим»

При связывании отношений необходимо следовать двум фундаментальным правилам целостности: первичный ключ не может быть пустым, и внешний ключ может быть либо пустым, либо равен значению первичного ключа, на который он ссылается. Только в этом случае можно добиться непротиворечивости представления данных.

3.1.4 Реляционные операции

Реляционная модель данных основана на применении реляционной алгебры Эдгара Кодда. В свою очередь, за реляционной алгеброй стоит теория множеств, в которой определено восемь простейших операций над множествами: выборка, проекция, пересечение, сложение, вычитание, умножение, деление и переименование.

С помощью этих операций можно строить более сложные операции, называемые объединениями, которые в языке структурированных запросов SQL реализуется с помощью инструкции SELECT. Результатом операции над отношением является представление, которое не сохраняется в базе данных, но в течение некоторого времени доступно для чтения.

Рассмотрим примеры основных операций над множествами.

Операция выборки выполняется над одним отношением. Результирующее представление будет содержать все исходные атрибуты, но, возможно, не все кортежи.

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56
Сидоров	Физика	35
Васильев	Информатика	80
Давыдов	Физика	75

Выборка →

Студент	Дисциплина
Иванов	Инфоматика
Петров	История
Сидоров	Физика
Васильев	Информатика
Давыдов	Физика

Рисунок 3.6 – Пример операции выборки

Операция проекции возвращает все кортежи исходного отношения, но, возможно, не все атрибуты.

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56
Сидоров	Физика	35
Васильев	Информатика	80
Давыдов	Физика	75

Проекция →

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Васильев	Информатика	80

Рисунок 3.7 – Пример операции проекции

Операция пересечения выполняется над двумя отношениями одинаковой структуры. В результате возвращаются только те кортежи, которые встречаются в обоих отношениях.

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56
Сидоров	Физика	35

Студент	Дисциплина	Рейтинг
Васильев	Физика	77
Петров	История	56
Иванов	Информатика	88

Пересечение →

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56

Рисунок 3.8 – Пример операции пересечения

Операция сложения также выполняется над двумя отношениями одинаковой структуры. При этом представление содержит все кортежи исходных отношений.

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56
Сидоров	Физика	35

Студент	Дисциплина	Рейтинг
Васильев	Физика	77
Петров	История	56
Иванов	Информатика	88

Сложение →

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56
Сидоров	Физика	35
Васильев	Физика	77

Рисунок 3.9 – Пример операции сложения

Операция вычитания возвращает кортежи первого отношения, отсутствующие во втором отношении.

Студент	Дисциплина	Рейтинг
Иванов	Инфоматика	88
Петров	История	56
Сидоров	Физика	35

Студент	Дисциплина	Рейтинг
Васильев	Физика	77
Петров	История	56
Иванов	Информатика	88

Вычитание →

Студент	Дисциплина	Рейтинг
Сидоров	Физика	35

Рисунок 3.10 – Пример операции вычитания

Операция умножения объединяет каждый кортеж первого отношения с каждым кортежем второго отношения. Эту операцию также называют декартовым произведением. Количество кортежей результирующего представления равно произведению числа кортежей исходных отношений.

Студент	Рейтинг
Иванов	88
Петров	56
Сидоров	35

Студент	Рейтинг
Васильев	80
Давыдов	75

Умножение →

Студент	Рейтинг	Студент	Рейтинг
Иванов	88	Васильев	80
Иванов	88	Давыдов	75
Петров	56	Васильев	80
Петров	56	Давыдов	75
Сидоров	35	Васильев	80
Сидоров	35	Давыдов	75

Рисунок 3.11 – Пример операции умножения

Операция деления выполняется над двумя отношениями, первое из которых состоит из двух атрибутов, а второе – из одного. Значения атрибута второго отношения сравниваются со значениями первого атрибута первого отношения, и если обнаруживаются совпадения, то соответствующие значения второго атрибута первого отношения включаются в представление.

Студент	Рейтинг
Иванов	85
Иванов	70
Иванов	69
Петров	85
Петров	75
Петров	70
Сидоров	77
Сидоров	69
Васильев	85
Давыдов	69

Студент
Иванов
Петров

Деление →

Рейтинг
85
70

Рисунок 3.12 – Пример операции деления

Операция переименования изменяет имя атрибута и является тривиальной операцией.

Студент	Рейтинг
Иванов	88
Петров	56
Сидоров	35
Васильев	80
Давыдов	75

Переименование →

Student	Rating
Иванов	88
Петров	56
Сидоров	35
Васильев	80
Давыдов	75

Рисунок 3.13 – Пример операции переименования

Перечисленные выше реляционные операции по отдельности применяются редко, однако их комбинации позволяют получать выборки данных практически любой сложности.

3.2 Правила Кодда

Для определения, является ли база данных реляционной или нет, Эдгар Кодд разработал двенадцать правил. Если база данных удовлетворяет всем этим правилам, то ее с уверенностью можно назвать реляционной. Рассмотрим правила Кодда более подробно.

Первое правило Кодда – данные в отношениях должны иметь явное представление. Это правило свидетельствует о том, что в реляционной базе данных должен выполняться принцип информационной неделимости, рассмотренный ранее.

Второе правило Кодда – к данным должен быть обеспечен гарантированный доступ. В этом случае в любой реляционной базе данных пользователь, зная имя отношения, значение первичного ключа и имя атрибута, должен получить доступ к необходимым данным.

Третье правило Кодда – в реляционной базе данных должна быть реализована обработка неопределенных значений. Это положение говорит о том, что в реляционных базах данных для всех типов данных должна поддерживаться работа с неопределенными значениями NULL.

Четвертое правило Кодда – описание базы данных должно осуществляться в терминах реляционной модели. Четвертое правило Кодда выдвигает дополнительное требование к описанию базы данных, которое должно храниться в виде отдельных связанных между собой отношений, а доступ к его содержимому должен осуществляться с помощью описанных выше реляционных операций.

Пятое правило Кодда – в реляционной базе данных должна быть обеспечена полнота подмножества языка. При этом язык манипулирования данными и язык определения данных должны быть необходимым и достаточным средством работы с данными.

Шестое правило Кодда – реляционная база данных должна поддерживать обновление представлений. В данном случае созданные посредством реляционных операций представления при изменении данных в исходных отношениях должны также обновляться.

Седьмое правило Кодда – в реляционной базе данных для манипулирования данными должен использоваться высокоуровневый язык. Синтаксис операций, применяемых к отношению в целом или к кортежам в частности, должен быть понятен любому пользователю.

Восьмое правило Кодда – в реляционной базе данных должна быть обеспечена физическая независимость данных. Выполнение этого правила приводит базу данных в соответствие модели ANSI/SPARC, в которой прикладные программы не должны зависеть от особенностей представления данных на физическом уровне.

Девятое правило Кодда – в реляционной базе данных должна быть реализована логическая независимость данных. Это правило говорит о том, что приложения реляционных баз данных не должны зависеть от логических ограничений, накладываемых на отношения.

Десятое правило Кодда – в реляционной базе данных должна быть обеспечена независимость контроля целостности. Следование этому правилу приводит к необходимости хранения средств поддержания целостности данных, таких как ог-

раничения на первичные ключи, внешние ключи, пределы допустимого диапазона значений данных, в отдельных отношениях базы данных.

Одиннадцатое правило Кодда – реляционная база данных должна обладать дистрибутивной независимостью данных. В этом случае к базе данных выдвигается требование переносимости на другие платформы и способности к распространению, в том числе в локальных и глобальных сетях.

Двенадцатое правило Кодда – реляционная база данных должна обеспечивать согласование языковых уровней. Последнее правило Кодда в основном направлено на согласование выполнения низкоуровневых операций, таких как резервное копирование и архивация данных, и высокоуровневых операций, например, выборка данных, разделение прав пользователей и привилегий доступа.

Привила Кодда являются в большей степени теоретическими требованиями, выдвигаемыми к построению реляционных баз данных. На практике ни одна база данных не реализует реляционную модель целиком. Разработчики баз данных руководствуются принципом разумной достаточности, разрабатывая систему, не противоречащую необходимым требованиям реляционных баз данных и отвечающую достаточным требованиям заказчика.

3.3 Нормализация реляционных баз данных

После статьи, посвященной реляционной алгебре, Эдгар Кодд в 1972 г. опубликовал работу под названием «Дальнейшая нормализация реляционной модели баз данных», в которой ввел термин «нормализация» и сформулировал три нормальные формы.

Нормализация – это метод организации реляционной базы данных с целью сокращения избыточности.

При разработке и проектировании любой реляционной базы данных одной из первых решается задача нормализации ее отношений, причем этот процесс является итерационным и заключается в последовательном переводе отношений из первой нормальной формы в нормальные формы более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивает функциональную зависимость отношений при сохранении свойств предшествующих нормальных форм.

Нормализация позволяет в полной мере реализовать преимущества реляционной модели, заставляет разработчика создавать больше отношений, равномернее распределяя в них информацию, что приводит к снижению избыточности и повышению целостности данных.

Всего выделяют пять нормальных форм. Рассмотрим каждую из них более подробно.

3.3.1 Первая нормальная форма

Первое правило реляционной модели состоит в том, что ни один столбец не должен содержать два или более значений.

Отношение представлено **в первой нормальной форме (1НФ)** тогда и только тогда, когда все его атрибуты являются простыми, то есть содержат только неделимые порции данных.

Приведем пример преобразования отношения в первую нормальную форму, используя таблицу, в которой хранятся данные о персональных компьютерах (рисунок 3.14).

information
$600 500 10 $6000
$850 750 5 $4250
$700 500 3 $2100
$450 600 8 $3600
$670 600 10 $6700

1НФ

price	speed	quantity	total
600	500	10	6000
850	750	5	4250
700	500	3	2100
450	600	8	3600
670	600	18	6700

Рисунок 3.14 – Преобразование отношения в первую нормальную форму

Как видно из рисунка 3.14 все данные о компьютерах, содержащиеся в исходном отношении, разбиты на четыре простых атрибута, содержащих неделимые порции данных.

Приведение отношения к первой нормальной форме упрощает разработку запросов и создание представлений, однако не исключает дублирования данных, что приводит потере целостности и увеличению объема занимаемой памяти. Для решения этих проблем необходимо использовать вторую нормальную форму.

3.3.2 Вторая нормальная форма

Для проведения дальнейшей нормализации отношений необходимо использовать понятие первичного ключа, так как вторая нормальная форма требует, чтобы все атрибуты зависели от первичного ключа.

Отношение представлено **во второй нормальной форме (2НФ)** тогда и только тогда, когда оно представлено в первой нормальной форме, и каждый неключевой атрибут полностью определяется первичным ключом.

price	speed	quantity	total
600	500	10	6000
850	750	5	4250
700	500	3	2100
450	600	8	3600
670	600	18	6700

2НФ

model	price	speed	quantity	total
1234	600	500	10	6000
1456	850	750	5	4250
1333	700	500	3	2100
1782	450	600	8	3600
1121	670	600	18	6700

Рисунок 3.15 – Преобразование отношения во вторую нормальную форму

Избыточность первой нормальной формы может послужить причиной возникновения аномалий вставки и удаления. Аномалии вставки возникают тогда, когда данные невозможно добавить в отношения до тех пор, пока они неполные, либо для этого требуется дополнительный просмотр отношения. Например, не имеет смысла хранить абстрактную информацию о компьютерах, если она не соответствует какой-либо модели.

Применение второй нормальной формы позволяет устранить аномалии вставки и удаления.

3.3.3 Третья нормальная форма

Применение первой и второй нормальных форм позволяет избавиться от явного дублирования данных, однако неявное дублирование данных все же имеет место. Решить эту проблему помогает использование третьей нормальной формы.

Отношение представлено **в третьей нормальной форме (3НФ)** тогда и только тогда, когда оно представлено во второй нормальной форме и все неключевые атрибуты отношения взаимно независимы и полностью зависят от первичного ключа.

model	price	speed	quantity	total
1234	600	500	10	6000
1456	850	750	5	4250
1333	700	500	3	2100
1782	450	600	8	3600
1121	670	600	18	6700

3НФ

model	price	speed	quantity
1234	600	500	10
1456	850	750	5
1333	700	500	3
1782	450	600	8
1121	670	600	18

Рисунок 3.16 – Преобразование отношения в третью нормальную форму

Если в отношении имеется зависимость атрибутов составного ключа от неключевых атрибутов, то необходимо перейти к усиленной третьей нормальной форме, которая называется нормальная форма Бойса-Кодда.

Отношение представлено **в нормальной форме Бойса-Кодда (БКНФ)** тогда и только тогда, когда оно находится в третьей нормальной форме и в нем отсутствуют зависимости атрибутов составного ключа от неключевых атрибутов.

Приведение отношений в третью нормальную форму является достаточным для решения практических задач, и на этом в большинстве случаев процесс проектирования реляционной базы данных заканчивается.

3.3.4 Четвертая нормальная форма

В некоторых случаях в отношениях появляется многозначная зависимость данных, заключающаяся в зависимости атрибутов составного ключа. Удаление или добавление кортежей приводит к обработке всех кортежей отношения, что ве-

дет к избыточности. Для исключения многозначных зависимостей необходимо использовать четвертую нормальную форму.

Отношение представлено **в четвертой нормальной форме (4НФ)** тогда и только тогда, когда оно находится в нормальной форме Бойса-Кодда и существует многозначная зависимость первичного ключа от другого атрибута, а все остальные атрибуты функционального зависят от первичного ключа.

3.3.5 Пятая нормальная форма

Результатом нормализации всех предыдущих операций были два новых отношения. Иногда это сделать не удается, либо получаемые отношения заведомо имеют нежелательные свойства. В этом случае выполняют декомпозицию исходного отношения на отношения, количество которых превышает два. Однако такая декомпозиция не должна приводить к потере данных и появлению дополнительных фиктивных кортежей. Именно для этого разработана пятая нормальная форма.

Отношение находиться **в пятой нормальной форме (5НФ)** тогда и только тогда, когда оно представлено в четвертой нормальной форме и не содержит зависимостей соединения.

Нужно отметить, что глубокое понимание процессов, происходящих при преобразовании в четвертую и пятую нормальные формы, требует привлечения математического аппарата теории множеств, что выходит за пределы изучения данного курса. В целом нужно отметить, что четвертая и пятая нормальные формы в большей степени представляют теоретический интерес, нежели практический.

3.3.6 Денормализация баз данных

При проектировании базы данных, удовлетворяющей всем нормальным формам, появляется переизбыток отношений, так как переход к очередной нормальной форме ведет к разделению отношений. При этом устраняется избыточность данных, однако появляется избыточность внутренних связей между отношениями. Кроме того, возникают дополнительные затраты на выполнение операций объединения, что значительно влияет на производительность.

Устранение избыточности не обязательно означает повышение производительности. Накладные расходы на выполнение операций объединения значительны, поэтому разработчики иногда сознательно идут на нарушение правил нормализации. Процесс нарушения нормальных форм называется **денормализацией**.

Денормализация вызвана несколькими причинами, основными из которых являются время выполнения запросов, время проведения обновлений, общий объем хранилища данных, аномалии удаления, вызывающие потерю целостности данных.

Тезаурус

Отношение, кортеж, атрибут, сущность, домен, схема отношений, ключ, первичный ключ, внешний ключ, индекс, подчиненность отношений, выборка, представление, правила Кодда, нормализация, нормальная форма, денормализация.

Контрольные вопросы

1) Назовите альтернативные варианты реляционных терминов.
2) В чем заключается принцип информационной неделимости?
3) В каких случаях применяется отношение «один – ко многим»?
4) Перечислите простейшие реляционные операции.
5) Приведите пример операции деления множеств.
6) Какие требования к базе данных выдвигает принцип дистрибутивной независимости данных, сформулированный Коддом?
7) Зачем в реляционной модели используются неопределенные значения?
8) Каков процесс возникновения аномалий вставки и удаления?
9) В чем отличие третьей нормальной формы от нормальной формы Бойса-Кодда?
10) Каковы причины применения денормализации?

Список рекомендуемой литературы

1) Агальцов, В. П. Базы данных. Локальные базы данных [Текст] / В. П. Агальцов. – М. : Форум, 2009. – 352 с.
2) Кириллов, В. В. Введение в реляционные базы данных [Текст] / В. В. Кириллов, Г. Ю. Громов. – СПб. : BHV, 2009. – 464 с.
3) Кренке, Д. Теория и практика построения баз данных [Текст] / Д. Кренке. – СПб. : Питер, 2003. – 800 с.
4) Кузнецов, С. Д. Основы баз данных [Текст] : учебное пособие / С. Д. Кузнецов. – М. : Бином, 2007. – 484 с.
5) Райордан, Р. М. Основы реляционных баз данных [Текст] / Р. М. Райордан. – М. : Русская Редакция, 2001. – 384 с.
6) Советов, Б. Я. Базы данных: теория и практика [Текст] / Б. Я. Советов, В. Д. Чертовской. – М. : Высшая школа, 2005. – 463 с.
7) Харрингтон, Д. Проектирование реляционных баз данных [Текст] / Д. Харрингтон. – М. : Лори, 2006. – 230 с.

ГЛАВА 4
ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ РАБОТЫ БАЗ ДАННЫХ

В настоящей главе рассматриваются вопросы, связанные с методами и средствами оптимизации работы баз данных, реализацией параллельных вычислений, а также организацией безопасности информации.

4.1 Параллельные вычисления

В настоящее время очень популярны Интернет-магазины и бухгалтерские многопользовательские системы, суть работы которых заключается в следующем. Определенный вид товара поступает на склад предприятия, после чего он продается покупателю, причем продажа не должна произойти ранее поступления, и один и тот же товар не должен быть продан разным покупателям. Такой процесс называется **параллелизмом**. От успешности реализации параллельных вычислений зависит эффективность работы системы в целом.

Параллелизм является сложной проблемой для систем управления базами данных, которые должны справляться не только с множественными запросами на подключение, но и планировать одновременный доступ к данным. Для решения таких задач используются транзакции.

4.1.1 Понятие транзакции

Транзакцией называется последовательность операций, производимых над базой данных и переводящих базу данных из одного непротиворечивого состояния в другое непротиворечивое состояние.

В конце транзакции происходит либо ее отмена, либо завершение. Отмена транзакции называется откатом, завершение транзакции называется фиксацией.

Транзакция должна обладать четырьмя свойствами: атомарностью, согласованностью, изолированностью и устойчивостью.

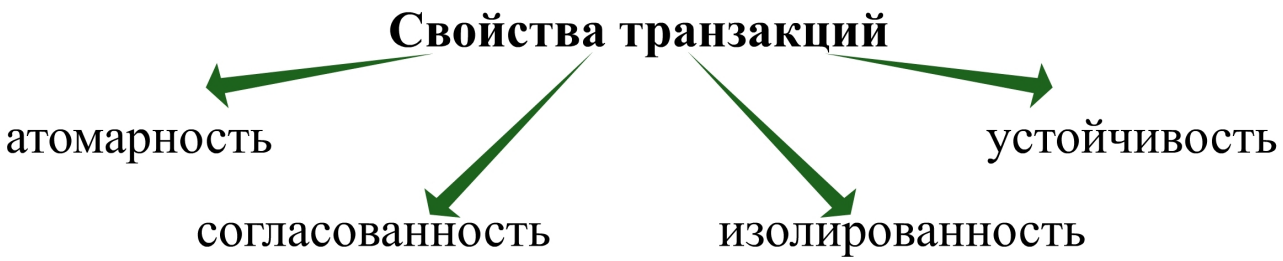


Рисунок 4.1 – Свойства транзакции

Свойство атомарности означает следование принципу неделимости транзакций. Данный принцип заключается в том, что все инструкции, составляющие транзакцию, обязательно выполняются, иначе не выполняется ни одна из них. Никаких промежуточных состояний не существует.

Свойство согласованности гарантирует, что по мере выполнения транзакции данные переходят из одного согласованного состояния в другое, при этом не нарушается взаимная согласованность данных в базе данных.

Свойство изоляции заключается в организации последовательной обработки транзакций при доступе к базе данных, при этом у пользователя появляется впечатление, что все транзакции выполняются параллельно.

Свойство устойчивости означает, что при успешном завершении транзакции в базу данных вносятся постоянные, неотменяемые изменения. База данных, поддерживаемая выполнение устойчивых транзакций, способна выдержать внезапную аварию, например, сбой питания, и при этом остаться согласованной.

Транзакции реализуются путем ведения журнала всех изменений, вносимых в базу данных в ходе выполнения каждой транзакции. По журналу можно восстановить согласованное состояние базы данных в случае сбоя.

Существует несколько видов сбоя, классификация которых приведена на рисунке 4.2.

Рисунок 4.2 – Классификация видов сбоя баз данных

Выделяют два вида аппаратных сбоев. Мягкие сбои возникают при внезапной остановке работы компьютера, например, при аварийном выключении питания. Жесткие сбои проявляют себя при выходе компьютера из строя и влекут потерю информации на носителях внешней памяти.

Среди программных видов сбоев следует отметить аварийное завершение работы системы управления базами данных и аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной, и требуется ликвидировать ее последствия.

Поддержание надежности хранения данных является приоритетной задачей работы системы управления базами данных. Для этого в базе данных необходимо хранить избыточную информацию для восстановления, причем эта информация

должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточности является ведение журнала изменений базы данных.

Журналом транзакций называется механизм сохранения промежуточных состояний, подтверждения или отката транзакции, реализованный в системах управления базами данных.

Во многих случаях придерживаются стратегии «упреждающей» записи в журнал транзакций (так называемого протокола Write Ahead Log – WAL), заключающейся в том, что запись об изменении любого объекта базы данных должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части базы данных.

Следует отметить, что использование транзакции при оптимизации баз данных снижает производительность работы, однако повышает целостность базы данных. Отсутствие транзакций приводит к значительному росту производительности, но страдает целостность, однако, существуют и промежуточные варианты, называемые уровнями изоляции транзакций, которые позволяют найти компромисс между двумя этими противоречивыми требованиями.

4.1.2 Уровни изоляции

Принято выделять четыре уровня выполнения транзакций: уровень сериализации транзакций (SERIALIZIBLE), уровень достоверного чтения (REPEATABLE READ), уровень подтвержденного чтения (READ COMMITTED), уровень неподтвержденного чтения (READ UNCOMMITTED). Самый высший уровень выполнения транзакций направлен на сохранение целостности данных и решения проблем, связанных с появлением промежуточных данных, несогласованных данных и строк-призраков, а самый низший уровень исключительно нацелен на повышение скорости выполнения запросов. Рассмотрим каждый из этих уровней более подробно.

Промежуточные данные появляются тогда, когда незавершенная транзакция модифицирует кортеж отношения, а другая транзакция читает этот кортеж. Если первая транзакция будет отменена, то окажется, что вторая транзакция получила данные, которые никогда не существовали. Такая ситуация появляется на уровне неподтвержденного чтения.

На уровне подтвержденного чтения транзакции могут читать только подтвержденные данные. Однако это не решает проблему несогласованных данных. Например, в ходе транзакции выполняется запрос, определяющий среднее значение по какому-либо числовому атрибуту отношения. По завершении этого запроса другая транзакция, работающая с тем же отношением, удаляет или добавляет кортежи. Если первая транзакция повторно выполнит свой запрос, то она получит другие результаты.

Проблема несогласованных данных решается на уровне достоверного чтения. В этом режиме кортежи, к которым транзакция обращается для чтения или записи, блокируются. В данном случае возникает проблема, связанная с появлением кортежей-призраков.

Транзакция может заблокировать все кортежи, с которыми ведется работа, но она не может помешать другой транзакции добавить кортеж в то же самое отношение. Если в ходе транзакции вводятся два запроса на выборку, а между ними вторая транзакция добавляет в отношение новый кортеж, то этот кортеж станет «призраком», так как он появляется в ходе одной и той же транзакции.

На уровне сериализации транзакции принудительно выполняются одна за другой в порядке приоритета. Если две транзакции попытаются обновить один и тот же кортеж, то одна из транзакций будет объявлена проигравшей в тупиковой ситуации и отменена.

На любом уровне ниже уровня сериализации появляется риск нарушения целостности данных. Однако, если в рамках поставленных задач требуется увеличить скорость выполнения запросов, то целостность данных уходит на второй план, и можно применять один из трех других уровней выполнения транзакций.

4.1.3 Выполнение транзакций

Многие системы управления базами данных по умолчанию работают в режиме автоматического завершения транзакций, в этом случае каждый запрос является самостоятельной транзакцией, которая после получения результатов запроса немедленно завершается.

Однако для обеспечения одновременного доступа к данным используются механизмы блокировки базы данных. Система управления базами данных может блокировать в случае необходимости отношения целиком, кортежи, атрибуты или страницы, которые являются произвольным блоком данных, связанным с отношением.

Также для выполнения транзакций в некоторых системах управления базами данных используется механизм составления последовательности выполнения транзакций, представляющий собой счетчик для создания уникальных числовых идентификаторов. В этом случае в базе данных никакие потоки выполнения транзакций не получат одинаковые идентификаторы, следовательно, не помешают работе друг друга.

Проектировщики баз данных могут создавать последовательности и сами с помощью первичных ключей с автоматическим увеличением значения при добавлении кортежа.

4.1.4 Обработка транзакций

Отслеживать уровень выполнения транзакции разработчику необходимо при проектировании специализированной базы данных, во многих других случаях имеет смысл использовать уже готовые и проработанные системы обработки транзакций, например, OLTP-системы, OLAP-системы или мониторы транзакций. Рассмотрим их более подробно.

4.1.4.1 OLTP-системы

В случае использования в базе данных сильно нормализованных моделей данных принято применять так называемые OLTP-приложения (On-Line Transaction Processing – оперативная обработка транзакций).

Основной функцией таких систем является выполнение большого количества коротких транзакций, при этом транзакции выглядят относительно просто, например, операция чтения или записи, и время ожидания запросов не превышает нескольких секунд.

Однако OLTP-системы имеют ряд недостатков. В таких системах транзакций очень много, выполняются они одновременно, и при ошибке транзакция должна целиком откатиться.

Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора данных из различных отношений. При этом большая часть запросов известна заранее еще на этапе проектирования системы. Таким образом, критическими для OLTP-приложений являются скорость и надежность выполнения коротких операций обновления данных. OLTP-системы требуют защиты от несанкционированного доступа, от нарушения целостности, аппаратных и программных сбоев.

4.1.4.2 OLAP-системы

В базах данных, использующих принципы построения систем поддержки принятия решений, хранилищ данных, систем интеллектуального анализа данных, применяются OLAP-приложения (On-Line Analitical Processing – оперативная аналитическая обработка данных).

Такие системы характеризуются следующими признаками. Добавление данных происходит редко и в основном крупными блоками. Данные, добавленные в систему, обычно никогда не удаляются. В таких системах перед загрузкой данные проходят различные процедуры обработки. Запросы к базе данных являются достаточно сложными. При всем при этом скорость выполнения запросов важна, но не критична.

Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов и используют многомерные модели данных. Оперативность обработки больших объемов данных в OLAP достигается за счет применения мощной многопроцессорной вычислительной техники, сложных методов анализа и специальных хранилищ данных, накапливающих информацию из разных источников за большой период времени и обеспечивающих быстрый доступ к ней. В большинстве случаев сложный аналитический запрос невозможно сформулировать в терминах стандартного языка запросов SQL, и приходится применять специализированные языки, ориентированные на аналитическую обработку данных.

4.1.4.3 Мониторы транзакций

С ростом сложности распределенных вычислительных систем возникают проблемы эффективного использования их ресурсов. Для решения этих проблем в состав распределенных OLTP-систем вводят дополнительный компонент – монитор транзакций.

Мониторы транзакций выполняют две основные функции в системах управления базами данных: динамическое распределение запросов в системе и оптимизацию числа выполняемых серверных приложений. Первая функция направлена на оптимальное распределение вычислительных ресурсов между несколькими серверами при обработке большого количества запросов, вторая функция направлена на решение задачи запуска необходимого и достаточного количества серверных приложений для обработки запросов с обеспечением максимальной скорости обработки транзакций.

Если в системе используется монитор транзакций, то со стороны разработчика не требуется дополнительных затрат для поддержания контроля целостности данных, и он может сосредоточиться на решении других задач.

4.2 Оптимизация баз данных

Оптимизация баз данных представляет собой процесс настройки системы, направленный на повышение скорости ее работы или сокращение объема используемой памяти. В большинстве случаев успех при оптимизации баз данных достигается тогда, когда накоплен достаточный опыт проектирования и сопровождения таких систем. В конечном итоге оптимизация направлена на экономию денежных средств, связанных с эксплуатацией и сопровождением базы данных, следовательно, финансовые затраты на оптимизацию не должны превышать экономической эффективности от внедрения средств оптимизации.

Оптимизация начинается с организационных мер, направленных на обновление аппаратной части, обновление операционной системы и обновление непосредственно системы управления базами данных (рисунок 4.3).

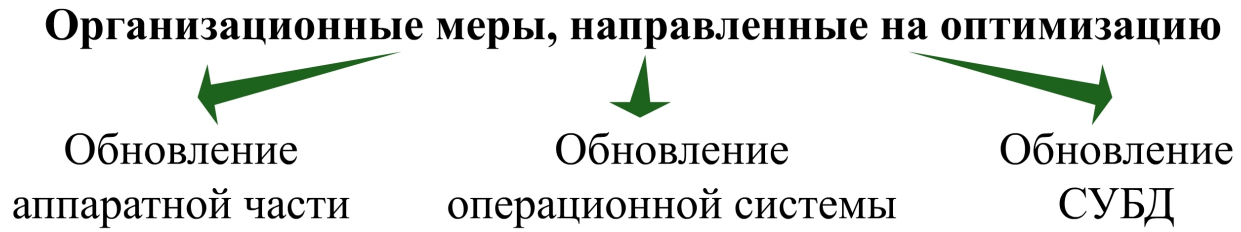


Рисунок 4.3 – Организационные меры при оптимизации базы данных

Наибольший эффект приносит обновление аппаратной части, при этом именно эта организационная мера может оказаться самым дешевым вариантом. Гордон Мур установил, что вычислительные мощности удваиваются каждые полтора года, несмотря на столь стремительный рост производительности, удельная стоимость вычислительных средств неуклонно снижается, что и гарантирует достижение необходимого результата.

Следующим этапом организационных мер, направленных на оптимизацию базы данных, является обновление операционной системы. В зависимости от целей и задач, которые ставятся перед базой данных, можно использовать либо бесплатные облегченные операционные системы, либо коммерческие операционные системы.

Обновлению подлежит и сама система управления базами данных. Постоянно выпускаются новые версии программных продуктов такого типа, которые решают либо новые задачи, либо старые задачи, на новом уровне. При этом в новых версиях систем управления базами данных исправлены ошибки, которые также могут существенно влиять на эффективность работы.

Однако, не всегда получается достичь должного эффекта с использованием организационных мер, тогда приходится применять непосредственно методы оптимизации баз данных. Классификация методов оптимизации баз данных приведена на рисунке 4.4. Рассмотрим каждый из них более подробно.

Рисунок 4.4 – Методы оптимизации баз данных

Предварительным этапом оптимизации баз данных является измерение производительности. Практически все производители систем управления базами данных, а также альтернативные разработчики, предоставляют комплекс тестов производительности, включающих в себя проверку нескольких стандартных аспектов функционирования баз данных, например, скорости выполнения запросов, операций чтения и записи, а также советы по устранению недочетов, выявленных в процессе тестирования системы.

Такие тесты отражают лишь относительную производительность сервера. С их помощью можно узнать, насколько возрастет скорость его работы при изменении рекомендуемых настроек, они лишь косвенно помогают в оптимизации баз данных.

Оптимизация базы данных начинается с оптимизации проекта. Основной целью данного метода является определение приемлемого уровня избыточности данных и денормализация некоторых отношений.

Для этого процесса характерно создание итоговых данных. Например, можно добавлять к отношению атрибут, хранящий результаты вычислений по другим атрибутам, иногда применяют технологию создания не только итоговых атрибутов, а целых отношений, причем, если такие отношения хранят часто изменяемую информацию, можно делать их резидентными.

Также особое внимание при оптимизации проекта уделяют определению типов данных. Например, иногда целесообразно не использовать тип переменной длины, поддерживаемый многими современными системами управления базами данных. Как правило, в них хранятся видеофильмы, звукозаписи, изображения. Более эффективно с точки зрения оптимизации проекта запоминать лишь путь до таких файлов, а хранить их отдельно от базы данных.

Оптимизация приложений, работающих с базой данных, в основном заключается в использовании технологии кэширования. Для быстрого подключения к базе данных можно кэшировать соединения с базой данных, можно кэшировать наиболее часто используемые данные из отношений, либо перед выполнением вставки кортежей в отношение, чтобы затем произвести выполнение этой операции в пакетном режиме.

В случае использования в качестве языка запросов SQL оптимизировать запрос довольно сложно, хотя различные подходы к реализации одной и той же задачи могут сэкономить процессорное время. При проектировании запросов необходимо особое внимание уделять работе с индексами, так как они позволят значительно повысить скорость работы приложения. Если подобный индекс еще не существовал, то при необходимости его нужно создать, так как во многих проектах приоритетной задачей является увеличение скорости работы приложения, а не экономия дискового пространства.

Кроме оптимизации запросов можно применять методы оптимизации других инструкций, например, инструкций вставки и удаления. Во избежание затрат вре-

мени на анализ инструкции можно воспользоваться преимуществами значений по умолчанию: вместо того чтобы указывать значения всех атрибутов, можно задать лишь те, которые отличаются от стандартных установок. Для того, чтобы не дублировать синтаксис однотипных операций, необходимо использовать многострочные инструкции. Также можно воспользоваться командами, позволяющими помещать инструкции в очередь без блокирования клиентского приложения, что повышает его оперативность.

Периодически для ускорения выполнения запросов необходимо проводить оптимизацию отношений. Во-первых, можно устранять пустые промежутки, оставшиеся после удаления кортежей. Во-вторых, можно соединять распределенные фрагменты отношений с динамическими кортежами. В-третьих, можно сортировать отношения и индексы. В случае если отношение в течение времени практически не меняется и дисковое пространство ограничено, можно архивировать отношение, однако после этого оно доступно лишь для чтения.

Распределение ресурсов сервера, а в основном оперативной памяти, также влияет на производительность баз данных. Увеличение объема памяти способствует ускорению работы системы управления базами данных, так как в оперативной памяти она хранит представления и временные отношения. Существует множество образцов конфигурационных файлов с различными вариантами настроек, касающихся использования памяти сервера, практически для всех баз данных и операционных систем. Перебор наиболее подходящих конфигураций сервера в совокупности с тестирование системы может принести положительные результат при проведении оптимизации баз данных.

Необходимость в компиляции при оптимизации баз данных возникает в том случае, когда для используемой платформы нельзя найти скомпилированную версию системы управления базами данных, либо используемые библиотеки или компиляторы морально устарели. Инструкций по перекомпиляции на информационных ресурсах разработчиков баз данных достаточно много, однако этот метод оптимизации является крайним и применяется в том случае, когда все другие варианты оптимизации не принесли приемлемого результата.

4.3 Безопасность баз данных

Для организации безопасности баз данных недостаточно организовать охрану помещений и установить программные средства защиты. В таком случае требуется комплексный подход, предполагающий организацию защиты системы управления базами данных в соответствии с ее структурой, задачами и возможностями защиты.

Систему безопасности баз данных нужно планировать с начала ее разработки и выполнять на всех этапах жизненного цикла. На практике создание системы защиты часто начинается в процессе эксплуатации базы данных. Система безо-

пасности должна быть многоуровневой, адаптируемой к новым условиям функционирования.

Для эффективного построения системы безопасности необходимо, во-первых, выделить уязвимые элементы системы управления базами данных, во-вторых, выявить угрозы для выделенных элементов, в-третьих, сформировать требования к системе безопасности и в конечно итоге выбрать методы и средства удовлетворения предъявленным требованиям.

Безопасность системы управления базами данных может быть нарушена вследствие реализации угроз, представляющих собой возможность преднамеренного или случайного действия, которое может принести к нарушению безопасности хранимой и обрабатываемой информации.

Основными видами угроз в системе управления базами данных являются несанкционированное или некорректное использование ресурсов, появление ошибок в программных и аппаратных средствах, хищение устройств носителей информации, вывод из строя средств хранения или передачи данных.

Существующие методы обеспечения безопасности баз данных можно разделить на четыре основных класса: физические, аппаратные, программные и организационные (рисунок 4.5). Рассмотрим каждый из этих методов более подробно.

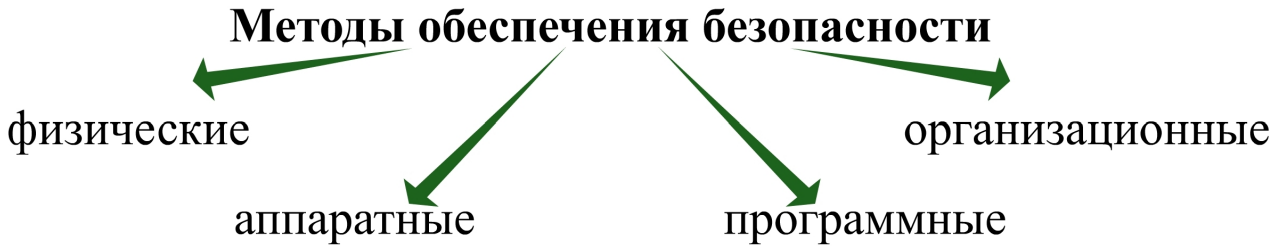


Рисунок 4.5 – Методы обеспечения безопасности баз данных

Физическая защита направлена на ограничение доступа посторонних лиц в помещение, где расположен сервер с установленной системой управления базами данных. Физические методы обеспечения безопасности включают средства охранной сигнализации, основанные на различных принципах.

Основными аппаратными средствами обеспечения безопасности являются средства защиты процессоров, например, проверка допустимости выдаваемых команд, устройств внешней памяти, каналов связи и систем электропитания.

К программным методам обеспечения безопасности баз данных относят применение встроенных средств операционных систем и антивирусных программ, использование специализированных коммерческих приложений и утилит.

Организационная защита включает в себя совокупность действий, связанных с разработкой и принятием законодательных документов, касающихся вопро-

сов защиты информации, а также с проведением пропагандистских мероприятий, призывающих работать с базами данных в рамках правового поля.

Однако наибольший интерес представляет синтез двух подходов, реализующий программно-аппаратные методы обеспечения защиты информации в базах данных. Программно-аппаратные средства обеспечения безопасности помогают защитить систему управления базами данных от внешних воздействий, так и предотвратить возникновение ошибок в самих приложениях, работающих с базами данных.

4.3.1 Защита баз данных от несанкционированного доступа

Для защиты от несанкционированного доступа с помощью программно-аппаратных средств используют технологию регистрации попыток доступа в систему со стороны пользователей и программ, а также гибкую политику быстрых уведомлений о таких попытках.

Защита от несанкционированного доступа со стороны пользователей в современных системах реализуется двумя способами: парольной защитой и путем идентификации и аутентификации.

Простейшая парольная защита является достаточно слабым средством обеспечения безопасности баз данных, особенно если пароль не шифруется. Основной ее недостаток состоит в том, что все пользователи, использующие одинаковый пароль, с точки зрения системы управления базами данных, идентичны. Неудобство парольной защиты для пользователя состоит в том, что при использовании простого и короткого пароля появляется угроза его подбора злоумышленниками, а если пароль сложный, то пользователь должен дополнительно его хранить на другом носителе и соответственно обеспечивать ему дополнительную защиту.

Более серьезный контроль доступа в систему получается, если каждого подключающегося пользователя сначала идентифицировать, затем аутентифицировать – убедиться, что это именно он, и при запросе ресурсов контролировать его полномочия.

Для аутентификации можно использовать личную информацию пользователя, электронные ключи, электронные жетоны, магнитные карточки, активные средства распознавания или биометрические средства. Наиболее перспективными являются биометрические средства защиты баз данных. К ним относятся контроль по отпечаткам пальцев, считывание сетчатки глаза, распознавание голоса, определение ДНК человека. Однако многие из этих методов проходят этапы доработки и апробации.

Одной из разновидностей несанкционированных программ являются компьютерные вирусы. Количество известных компьютерных вирусов постоянно возрастает, следовательно, и потенциальный ущерб от их работы пропорционально увеличивается. Поэтому проблема защиты баз данных от компьютерных вирусов

на всех стадиях их развития актуальна как никогда. Для этого в систему управления базами данных разработчики включают средства диагностирования состояния программно-аппаратных средств, локализации и удаления вирусов, устранения последствий их воздействия.

4.3.2 Защита баз данных от несанкционированного использования ресурсов

Обеспечение защиты от несанкционированного использования ресурсов требует применения средств мониторинга запросов защищаемых ресурсов и сигнализации при попытке их незаконного использования. К таким средствам относятся варианты защиты от копирования данных, исследования приложений, просмотра, модификации и удаления данных. Рассмотрим их более подробно.

Рисунок 4.6 – Примеры несанкционированного использования ресурсов

Для защиты системы управления базами данных от несанкционированного копирования в приложениях можно использовать привязку к аппаратной части, чтобы созданная копия была неработоспособна на другом компьютере.

При несанкционированном исследовании приложений применяются такие средства, которые не позволяют или затрудняют изучение системы защиты базы данных. Например, после нескольких неудачных попыток подключения к приложению, имеющему парольную защиту, необходимо блокировать дальнейшие попытки подключения к ней либо предусмотреть средства самоликвидации.

Самым эффективным средством защиты данных от просмотра является их шифрование. Несанкционированный просмотр данных при этом требует использования ключа шифрования, подбор которого даже при современном уровне развития информационных технологий представляет трудоемкую задачу.

Шифрование незаменимо для защиты информации от раскрытия ее содержания в системах управления базами данных, а также при ее передаче по линиям связи. Шифрование данных осуществляется в темпе поступления информации и в автономном режиме. Первый способ применяется в системах приема-передачи информации, а второй – для засекречивания информации.

Распространенным подходом к решению задачи защиты данных от модификации является вычисление контрольных сумм и их сравнение с эталоном.

Защитить данные от удаления можно путем предотвращения несанкционированных операций удаления в системе управления базами данных. Для этих целей необходимо разрабатывать дополнительные программные средства, которые параллельно с выполнением пользовательского приложения отслеживают поступившие запросы на удаление и в случае их несанкционированного выполнения отказывают в доступе к данным.

4.3.3 Защита баз данных от некорректного использования ресурсов

Защита от некорректного использования ресурсов выполняется операционной системой и предусматривает выделение клиентским приложениям изолированной области оперативной памяти и защиту системных областей внешней памяти. Однако многие разработчики систем управления базами данных идут далее и внедряют различные инструменты корректного использования прикладных ресурсов, например, документов, изображений, звуковых файлов.

4.3.4 Защита баз данных с помощью внесения избыточности

Очень эффективным способом восстановления баз данных после сбоев является избыточность. Избыточность бывает функциональная, информационная или структурная. В зависимости от целей защиты базы данных можно использовать один из представленных видов избыточности или их композицию.

Структурная избыточность означает резервирование аппаратных компонентов вычислительной системы на различных уровнях: дублирование серверов обработки информации, дублирование накопителей информации. При резервировании необходимо в первую очередь обеспечить стабильное и бесперебойное питание вычислительной системы.

Функциональное резервирование означает организацию вычислительного процесса, при котором дублируются функции управления, хранения и обработки информации. Например, запуск нескольких одинаковых клиентских приложений в многозадачной операционной системе является признаком функциональной избыточности.

Информационное резервирование используется для предотвращения полной потери информации и реализуется путем одноразового или периодического копирования и архивирования наиболее ценной информации. В случае сбоя или потери информации можно, используя резервную копию, восстановить необходимую информацию, важным аспектом в данном случае является периодичность. Если резервирование проводится часто, то может встать проблема неэффективного использования дискового пространства, если резервирование проводится редко, то могут увеличиться затраты на восстановление информации, которая накопилась в базе данных между моментом последнего сохранения и моментом сбоя. Следова-

тельно, необходимо проводить гибкую политику в области информационного резервирования, исходя из целей и задач, которые ставятся перед базой данных.

Тезаурус

Параллельные вычисления, транзакция, журнал транзакций, уровни изоляции, мониторы транзакций, оптимизация баз данных, идентификация, аутентификация, шифрование, избыточность.

Контрольные вопросы

1) Назовите и охарактеризуйте свойства транзакций.
2) Перечислите существующие виды сбоев баз данных.
3) В чем заключается стратегия упреждающей записи при ведении журнала транзакций?
4) Какие недостатки присущи уровню неподтвержденного чтения?
5) Назовите области применения OLTP-систем.
6) Какие организационные меры применяют при оптимизации баз данных?
7) В каких случаях необходимо осуществлять перекомпиляцию системы управления базами данных?
8) В чем суть программных методов обеспечения безопасности данных?
9) Какие технологии используются для защиты баз данных от несанкционированного доступа?
10) Зачем при защите информации в систему управления базами данных вносят дополнительную избыточность?

Список рекомендуемой литературы

1) Аткинсон, Л. MySQL Библиотека профессионала [Текст] / Л. Аткинсон. – М. : Вильямс, 2002. – 624 с.
2) Карпова, Т. С. Базы данных: модели, разработка, реализация [Текст] / Т. С. Карпова. – СПб. : Питер, 2002. – 304 с.
3) Савицкий, Н. И. Технологии организации, хранения и обработки данных [Текст] / Н. И. Савицкий. – М. : Инфра-М, 2001. – 232 с.
4) Советов, Б. Я. Базы данных: теория и практика [Текст] / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. – М. : Высшая школа, 2005. – 463 с.
5) Стивенс, Р. Программирование баз данных [Текст] / Р. Стивенс. – М. : Бином-Пресс, 2007. – 384 с.
6) Харрингтон, Д. Разработка баз данных [Текст] / Д. Харрингтон. – М. : ДМК, 2005. – 272 с.
7) Хомоненко, А. Д. Базы данных [Текст] : учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. – СПб. : Корона принт, 2004. – 736 с.

ГЛАВА 5
СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ SQL

В настоящей главе рассматриваются вопросы построения запросов к реляционным базам данных с помощью структурированного языка запросов SQL, а также примеры использования запросов.

5.1 Введение в SQL

Элегантность и независимость от специфики компьютерных технологий, а также поддержка лидерами промышленности в области технологии реляционных баз данных, сделало SQL основным стандартным языком обращения к базам данных.

Структурированный язык запросов SQL (Structured Query Language) является языком взаимодействия с базами данных, применяемым в большинстве реляционных системах управления базами данных. Основной стандарт языка был принят в 1992 г., и называется SQL2. Сегодня ведется работа над наиболее современным стандартом SQL3.

SQL представляет собой язык программирования четвертого поколения, так как он максимально приближен к человеческому языку. Одной из главных задач SQL является легкость синтаксиса языка, он должен быть понятным всем, начиная от пользователей и заканчивая администраторами, поэтому запросы читаются как обычные предложения.

Словарь SQL относительно невелик, а его команды являются словами английского языка. Обычно ключевые слова SQL записывают прописными буквами, чтобы отличать их от названий таблиц и столбцов. Часто для удобства восприятия инструкции SQL записывают в нескольких строках, что допускается синтаксическим анализатором.

Для удобства дальнейших рассуждений при описании реляционной модели данных будем использовать табличный вариант терминов: отношение – таблица, кортеж – строка, атрибут – столбец, так как именно такая терминология используется при программировании баз данных.

5.2 Оператор SELECT

Оператор SELECT осуществляет выборку из базы данных и имеет наиболее сложную структуру среди всех операторов языка SQL. Этот оператор наиболее востребован, следовательно, получение навыков и умений работы в частности с оператором SELECT гарантирует успех при овладении языка структурированных запросов SQL в целом.

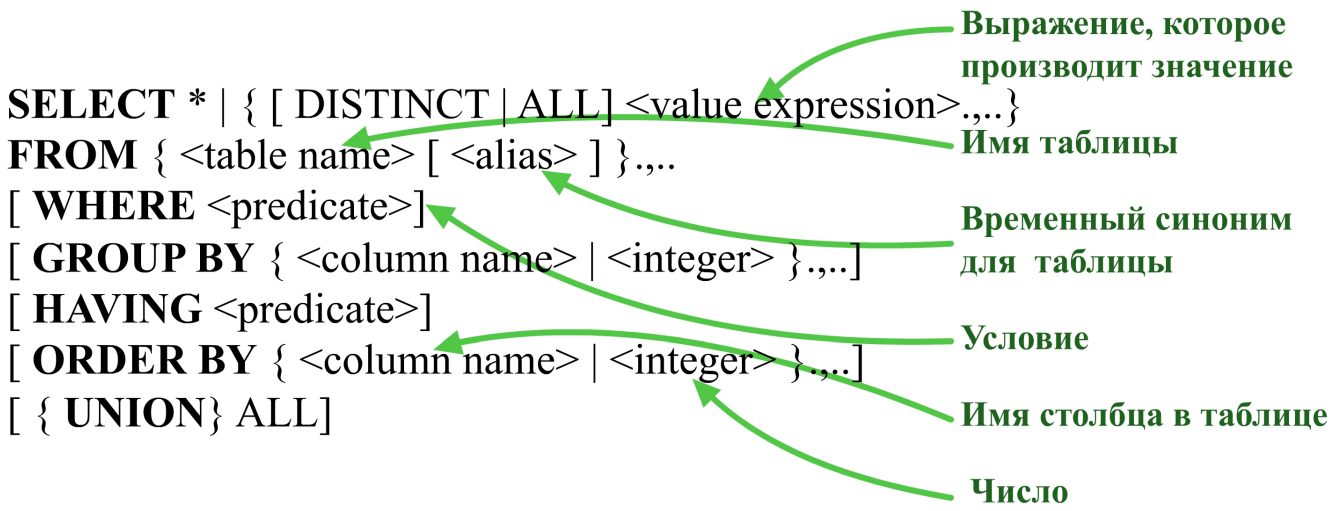


Рисунок 5.1 – Структура оператора SELECT

Рассмотрим синтаксис оператора SELECT (рисунок 5.1). В данном случае <value expression> – выражение, которое производит значение. Оно может включать в себя или содержать имена столбцов <column name> в таблице, <table name> является именем или синонимом таблицы, <alias> представляет собой временный синоним для таблицы, <predicate> – условие, которое может быть верным или неверным для каждой строки или комбинации строк таблицы в предложении FROM, <integer> – число с десятичной точкой.

Наиболее эффективным путем изучения языка структурированных запросов SQL является рассмотрение конкретных примеров использования различных конструкций.

Необходимо отметить, что структурированный язык запросов SQL имеет большое количество диалектов, которые имеют различные возможности для обработки данных и процедурные надстройки, например, Procedural SQL, Transact-SQL, PostgreSQL, однако все они лишь расширяют стандарт SQL2. Именно на этот стандарт ориентированные приведенные ниже примеры использования структурированного языка запросов.

В качестве предметной области для демонстрации работы запросов будем использовать учебную таблицу PC, в которой хранится информация о различных конфигурациях персональных компьютеров (рисунок 5.2). Данная таблица состоит из шести столбцов: price – в этом столбце хранится числовая информация о стоимости компьютера, speed – представляет собой столбец с числовой информацией о частоте работы микропроцессора, hd – содержит числовую информацию о размере жесткого диска, ram – в этом столбце хранится размер оперативной памяти компьютера в числовом виде, cd – столбец с текстовой информацией о скорости работы CD-ROM, model – столбец, в котором хранится уникальный код конфигурации компьютера в числовом виде.

price	speed	hd	ram	cd	model
600	500	20	256	48x	1234
850	750	40	512	50x	1456
700	500	30	512	52x	1333
450	600	10	256	50x	1782
670	600	20	256	24x	1121
920	800	160	1024	50x	2431

Рисунок 5.2 – Информация, содержащаяся в учебной таблице PC

5.2.1 Выборка данных

Простейший оператор SELECT, осуществляющий выборку всех строк из таблицы, можно записать в виде:

```
SELECT *
FROM PC
```

Чтобы упорядочить столбцы, их следует перечислить через запятую:

```
SELECT price, speed, hd, ram, cd, model
FROM PC
```

Результатом выполнения этих запросов станет выборка данных, соответствующая информации, которая представлена на рисунке 5.2.

Вертикальную выборку таблицы PC можно получить, если перечислить только необходимые столбцы (результат выполнения запроса представлен на рисунке 5.3, а):

```
SELECT hd, ram
FROM PC
```

Вертикальная выборка может содержать дубликаты строк. Если требуется получить уникальные строки, то можно использовать ключевое слово DISTINCT (результат выполнения запроса представлен на рисунке 5.3, б):

```
SELECT DISTINCT hd, ram
FROM PC
```

hd	ram
20	256
40	512
30	512
10	256
20	256
160	1024

а)

hd	ram
20	256
40	512
30	512
10	256
160	1024

б)

Рисунок 5.3 – Результат вертикальной выборки:
а) с дубликатами строк, б) без дубликатов строк

Чтобы упорядочить строки результирующего набора, можно выполнить сортировку по любому количеству столбцов, указанных в предложении SELECT. Для этого используется предложение ORDER BY:

```
SELECT DISTINCT hd, ram
FROM PC
ORDER BY ram DESC
```

Сортировку можно производить по возрастанию (параметр ASC по умолчанию) или по убыванию (параметр DESC). Пример сортировки по двум столбцам с использованием этих параметров выглядит следующим образом:

```
SELECT DISTINCT hd, ram
FROM PC
ORDER BY ram DESC, hd ASC
```

hd	ram
160	1024
40	512
30	512
20	256
10	256

а)

hd	ram
160	1024
30	512
40	512
10	256
20	256

б)

Рисунок 5.4 – Результат вертикальной выборки с сортировкой:
а) по одному столбцу, б) по двум столбцам

Горизонтальную выборку реализует предложение WHERE. При этом в результирующий набор попадут только те строки из источника данных, для каждой из которых значение предиката равно истине (TRUE). Приведем пример такого запроса (результат выполнения представлен на рисунке 5.5, а):

```
SELECT DISTINCT hd, ram
FROM PC
WHERE hd < 100
ORDER BY 2 DESC
```

hd	ram
40	512
30	512
20	256
10	256

а)

price	speed	model
700	500	1333
450	600	1782
670	600	1121

б)

Рисунок 5.5 – Результат горизонтальной выборки:
а) простой предикат, б) составной предикат

Дадим более точное определение понятию предикат. **Предикаты** представляют собой выражения, принимающие истинностное или ложное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов. Предикат в языке SQL может принимать одно из трех значений TRUE (истина), FALSE (ложь) или UNKNOWN (неизвестно).

Приведем примеры простых предикатов сравнения. Предикат price < 1000 означает, что в итоговую выборку попадут те модели, у которых цена меньше 1000. Предикат cd = '24x' означает, что в итоговую выборку попадут модели, содержащие 24-скоростные CD-ROM. Предикат hd < > 160 означает, что в итоговую выборку попадут модели, у которых размер жесткого диска не равен 160. Предикат ram – 128 > 0 означает, что в итоговую выборку попадут модели, у которых размер оперативной памяти превышает 128. Предикат price <= hd*2 означает, что в итоговую выборку попадут модели, у которых цена не превышает удвоенного размера жесткого диска. С помощью таких конструкций можно строить предикаты любой сложности, причем соединяются они между собой с помощью булевских операторов, например:

```
SELECT price, speed, model
FROM PC
WHERE speed >= 500 AND price < 800
```

Результат выполнения такого запроса представлен на рисунке 5.5, б. Иногда необходимо явно указать диапазон попадания значений, для этого используют предикат BETWEEN, который проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми служебным словом AND:

```
SELECT model, price
FROM PC
WHERE price BETWEEN 400 AND 600
```

Результат выполнения такого запроса представлен на рисунке 5.6, а.

Однако, не всегда удается определить один диапазон, в этом случае можно использовать предикат IN, который определяет, будет ли значение проверяемого выражения обнаружено в наборе значений, который либо явно определен, либо получен с помощью табличного подзапроса. Результат выполнения такого запроса представлен на рисунке 5.6, б. Такой вариант использования предиката IN позволяет решать широкий спектр задач:

```
SELECT model, speed, hd
FROM PC
WHERE hd IN (10, 20, 160)
```

model	price
1234	600
1782	450

а)

model	speed	hd
1234	500	20
1782	600	10
1121	600	20
2431	800	160

б)

Рисунок 5.6 – Результат горизонтальной выборки:
а) предикат BETWEEN, б) предикат IN

При выполнении различных операций имена столбцов, указанные в предложении SELECT, можно переименовать с помощью ключевого слова AS. Это делает результаты более читабельными:

```
SELECT ram*1024 AS Kb, 'Kb' AS Size
FROM PC
WHERE cd = '50x'
```

Результат выполнения такого запроса представлен на рисунке 5.7.

Kb	Size
524288	Kb
262144	Kb
1048576	Kb

Рисунок 5.7 – Результат переименования полей

Широко используется при формировании горизонтальных выборок предикат LIKE, который сравнивает строку, указанную в первом выражении с шаблоном, который определен во втором выражении. В образце разрешается использовать два трафаретных символа: символ подчеркивания «_», который можно использовать вместо любого единичного символа в проверяемом значении, и символ процента «%», который заменяет набор любых символов в проверяемом значении. Приведем пример таких запросов, результат выполнения которых приведен на рисунке 5.8:

```
SELECT model, speed, cd
FROM PC
WHERE cd LIKE '_0x'

SELECT model, speed, cd
FROM PC
WHERE cd LIKE '5%'
```

model	speed	cd
1456	750	50x
1782	600	50x
2431	800	50x

а)

model	price	cd
1456	850	50x
1333	700	52x
1782	450	50x
2431	920	50x

б)

Рисунок 5.8 – Результат горизонтальной выборки с использованием:
а) трафаретного символа «_», б) трафаретного символа «%»

Также довольно часто необходимо установить отсутствие или наличие значения в столбцах таблицы, для этих целей применяется предикат IS [NOT] NULL:

```
SELECT model, price
FROM PC
WHERE price IS NULL
```

5.2.2 Агрегатные функции

В SQL существует ряд агрегатных функций, каждая из которых оперирует совокупностью значений столбца некоторой таблицы и создает единственное значение.

Стандартом предусмотрены следующие агрегатные функции. Функция COUNT(*) возвращает количество строк источника данных. Функция COUNT(<имя поля>) возвращает количество значений в указанном столбце. Функция SUM(<имя поля>) возвращает сумму значений в указанном столбце. Функция AVG(<имя поля>) возвращает среднее значение в указанном столбце. Функция MIN(<имя поля>) возвращает минимальное значение в указанном столбце. Функция MAX(<имя поля>) возвращает максимальное значение в указанном столбце.

Функции COUNT, MIN и MAX применимы к любым типам данных, SUM и AVG применяются только для числовых полей. Рассмотрим более подробно использование каждой из этих функций.

Поиск максимального и минимального значений в выборке данных осуществляется с помощью агрегатных функций MAX и MIN:

```
SELECT
MIN(price) AS Min_price,
MAX(price) AS Max_price
FROM PC
```

Для получения количества строк в горизонтальной выборке данных используется функция COUNT:

```
SELECT COUNT(*) AS Qty
FROM PC
WHERE cd IN ('50x', '52x')
```

Для того, чтобы при получении результата агрегатных функций использовались только уникальные значения, при их аргументе можно также использовать параметр DISTINCT:

```
SELECT COUNT(DISTINCT cd) AS Qty
FROM PC
WHERE cd IN ('50x', '52x')
```

Поиск среднего значения в выборке данных осуществляется с помощью агрегатной функции AVG:

```
SELECT AVG(hd) AS Avg_hd
FROM PC
WHERE cd = '50x'
```

Результаты соответствующих запросов приведены на рисунке 5.9.

Min_price	Max_price
450	920

а)

Qty
4

б)

Qty
2

в)

Avg_hd
70

г)

Рисунок 5.9 – Результаты выполнения агрегатных функций:
а) поиск минимума и максимума, б) нахождение количества записей,
в) поиск количества уникальных записей, г) нахождение среднего значения

Существует несколько общих правил выполнения агрегатных функций, следование которым позволит избежать многих ошибок при разработке запросов:

1) если в результате выполнения запроса не получено ни одной строки, то исходные данные для вычисления любой из агрегатных функций отсутствуют. В этом случае результатом выполнения функций COUNT будет нуль, а результатом всех других функций – NULL;

2) аргумент агрегатной функции не может сам содержать агрегатные функции;

3) результатом выполнения функции COUNT является целое число. Другие агрегатные функции наследуют типы данных обрабатываемых значений;

4) если при выполнении функции SUM был получен результат, превышающий максимальное значение используемого типа данных, возникает ошибка.

5.2.3 Группировка данных

Значительно увеличивает ценность запросов возможность выполнения операции группировки данных, которая позволяет выполнять срезы информации в различных аспектах, это тем самым предоставляет дополнительный сервис пользователям коммерческих баз данных.

Предложение GROUP BY используется для определения групп выходных строк, к которым могут применяться агрегатные функции.

Если это предложение отсутствует, и используются агрегатные функции, то все столбцы с именами, упомянутыми в SELECT, должны быть включены в агрегатные функции. Иначе все столбцы списка SELECT, не вошедшие в агрегатные функции, должны быть указаны в предложении GROUP BY. Приведем пример использования группировки данных:

```
SELECT ram, COUNT(ram) AS Qty_ram, AVG(price) AS Avg_price
FROM PC
GROUP BY ram
```

Результат выполнения данного запроса приведен на рисунке 5.10.

ram	Qty_ram	Avg_price
256	3	573.3333
512	2	775
1024	1	920

Рисунок 5.10 – Результаты выполнения группировки данных

Функциональность группировки данных расширятся за счет применения предложения HAVING, которое используется после группировки для определения предиката, фильтрующего группы по значениям агрегатных функций. Это предложение необходимо для проверки значений, которые получены с помощью агрегатной функции:

```
SELECT ram, AVG(price) AS Avg_price
FROM PC
GROUP BY ram
HAVING ram IN (256, 512)
```

В предложении HAVING нельзя применять псевдоним (Avg_price), используемый для именования значений агрегатной функции:

```
SELECT ram, COUNT(ram) AS Qty_ram, AVG(price) AS Avg_price
FROM PC
GROUP BY ram
HAVING AVG(price) > 700
```

Результаты выполнения группировки данных с определением предиката представлены на рисунке 5.11.

ram	Avg_price
256	573.3333
512	775

а)

ram	Qty_ram	Avg_price
512	2	775
1024	1	920

б)

Рисунок 5.11 – Результаты выполнения группировки данных с определением предиката: а) простой предикат, б) сложный предикат

5.2.4 Условный оператор

В основу всех языком программирования изначально заложен принцип реакции программы на изменяющиеся внешние условия, который осуществляется с помощью условного оператора. SQL не стал исключением и реализует данный принцип с помощью двух альтернативных операторов CASE и IF. Рассмотрим их более подробно.

Оператор CASE в зависимости от указанных условий возвращает одно из множества возможных значений:

```
SELECT price, model,
CASE WHEN price >= 700 THEN 'Дорого'
ELSE 'Дешево' END AS comment
FROM PC
```

Функция IF возвращает разные значения в зависимости от того, истинным или ложным является проверяемое выражение. Результат проверки приводится к целому типу:

```
SELECT price, model,
IF(price >= 700, 'Дорого', 'Дешево') AS comment
FROM PC
```

Результат двух последних запросов дает одинаковую выборку данных, представленную на рисунке 5.12.

price	model	comment
600	1234	Дешево
850	1456	Дорого
700	1333	Дорого
450	1782	Дешево
670	1121	Дешево
920	2431	Дорого

Рисунок 5.12 – Результаты применения условного оператора

Функция IF является подобной оператору CASE, и может быть предпочтительней в простых случаях, однако не используется, когда необходимо применять подзапросы и сложные предикаты.

5.2.5 Соединение таблиц

Синтаксис оператора SELECT в предложении FROM допускает указание нескольких таблиц.

Рассмотрим примеры выполнения операций соединения таблиц на учебной базе данных, содержащей две таблицы, представленные на рисунке 5.13. Первая таблица PC состоит из двух числовых столбцов, содержащих информацию о моделях компьютеров (model) и значениях внешнего ключа (code), соответствующие их типу. Вторая таблица Sale содержит один числовой столбец (type), в котором хранится значение первичного ключа, и один текстовый столбец (comment), в котором хранится описание типа.

model	code
1234	1
1456	2
1333	1
1782	1
2212	4

а)

type	comment
1	продажа
2	резерв
3	заказ

б)

Рисунок 5.13 – Выборка данных из: а) таблицы PC б) таблицы Sale

Простое перечисление таблиц практически не используется, так как оно соответствует реляционной операции декартова произведения:

SELECT model, comment
FROM PC, Sale

Результат выполнения запроса, реализующего операцию декартова произведения, приведен на рисунке 5.14.

model	comment	model	comment	model	comment
1234	продажа	1782	резерв	1234	заказ
1456	продажа	2212	резерв	1456	заказ
1333	продажа	1234	резерв	1333	заказ
1782	продажа	1456	резерв	1782	заказ
2212	продажа	1333	резерв	2212	заказ

Рисунок 5.14 – Операция декартова произведения таблиц

Перечисление таблиц используется совместно с условием соединения строк из разных таблиц, указываемым в предложении WHERE:

```
SELECT model, comment
FROM PC, Sale
WHERE code = type
```

В предложении FROM может быть указана явная операция соединения двух и более таблиц. Соединение бывает либо внутренним (INNER), либо одним из внешних (OUTER). Внутреннее соединение INNER JOIN означает, что в результирующий набор попадут только те соединения строк таблиц, для которых значение предиката равно TRUE. Данное соединение таблиц аналогично использованию предложения WHERE:

```
SELECT model, comment
FROM PC
INNER JOIN Sale ON PC.code = Sale.type
ORDER BY model
```

Внешнее соединение LEFT JOIN означает, что помимо строк, для которых выполняется условие предиката, в результирующий набор попадут все остальные строки из первой таблицы (левой). При этом отсутствующие значения столбцов из правой таблицы будут заполнены NULL-значениями:

```
SELECT model, comment
FROM PC
LEFT JOIN Sale ON PC.code = Sale.type
ORDER BY model
```

Соединение RIGHT JOIN обратно соединению LEFT JOIN, то есть в результирующий набор попадут все строки из второй таблицы, которые будут соединяться только с теми строками из первой таблицы, для которых выполняется условие соединения.

При полном соединении (FULL JOIN) в результирующую таблицу попадут не только те строки, которые имеют одинаковые значения в сопоставляемых столбцах, но и все остальные строки исходных таблиц, не имеющие соответствующих значений в другой таблице. В этих строках все столбцы той таблицы, в которой не было найдено соответствия, заполняются NULL-значениями. Полное соединение равносильно комбинации левого и правого внешних соединений:

```
SELECT model, comment
FROM PC
FULL JOIN Sale ON PC.code = Sale.type
ORDER BY model
```

Результаты выполнения всех трех запросов по соединению таблиц приведены соответственно на рисунке 5.15.

model	comment
1234	продажа
1456	резерв
1333	продажа
1782	продажа

а)

model	comment
1234	продажа
1456	резерв
1333	продажа
1782	продажа
2212	*NULL*

б)

model	comment
1234	продажа
1456	резерв
1333	продажа
1782	продажа
2212	*NULL*
NULL	заказ

в)

Рисунок 5.15 – Операция явного соединения таблиц:
а) внутреннее соединение, б) левое соединение, в) полное соединение

Необходимо отметить, что иногда в предложении FROM требуется указать одну и ту же таблицу несколько раз. В этом случае обязательным является переименование:

```
SELECT DISTINCT A.model AS m1, B.model AS m2
FROM PC AS A, PC AS B
WHERE ABS(A.price – B.price) <= 100 AND A.model < B.model
```

Результат выполнения данного запроса, который выдает пары моделей персональных компьютеров, разница в стоимости которых не превышает 100, представлен на рисунке 5.16.

m1	m2
1234	1333
1121	1234
1456	2431
1121	1333

Рисунок 5.16 – Результат выполнения операции переименования таблиц

Переименование также требуется, если в предложении FROM используется подзапрос, однако такой вариант будет рассмотрен далее.

5.2.6 Операции над множествами

Декартово произведение является одной из главных операций над множествами, она реализуется перечислением через запятую табличных выражений в предложении FROM, такая форма записи использовалась ранее. Однако, имеется еще один способ формирования декартова произведения, который дает результат, аналогичный представленному на рисунке 5.14:

```
SELECT model, comment
FROM PC
CROSS JOIN Sale
```

В чистом виде декартово произведение практически не используется, наиболее популярной для решения прикладных задач является операция объединения. Оператор UNION объединяет выходные строки каждого из запросов в один набор. Если определен параметр ALL, то сохраняются все дубликаты выходных строк, иначе остаются только уникальные строки:

```
SELECT model, speed
FROM PC
WHERE price > 800
UNION ALL
SELECT model, speed
FROM PC
WHERE cd = '50x'
```

Результат объединения двух запросов представлен на рисунке 5.17.

model	speed
1456	750
2431	800
1456	750
1782	600
2431	800

Рисунок 5.17 – Результат выполнения операции объединения выборок

Использование при объединении запросов оператора UNION требует соблюдения следующих условий:

1) количество выходных столбцов каждого из запросов одинаково;

2) выходные столбцы каждого из запросов должны быть одного типа;

3) в выборке используются имена столбцов, заданные в первом запросе;

4) операция сортировки применяется к результату соединения.

В стандарте SQL имеются предложения оператора SELECT для выполнения операций пересечения и разности запросов: INTERSECT и EXCEPT, которые работают аналогично UNION. Однако эти команды поддерживаются далеко не во всех системах управления базами данных. Более универсальным является подход реализации операций пересечения и разности с помощью предиката существования EXISTS. Предикат EXISTS принимает значение TRUE, если подзапрос возвращает любое количество строк, иначе его значение равно FALSE, и для NOT EXISTS наоборот.

Рассмотрим пример, иллюстрирующий операцию пересечения двух множеств:

```
SELECT DISTINCT model, ram, hd
FROM PC AS Comp
WHERE ram < 1024 AND EXISTS
(SELECT model FROM PC
WHERE hd < Comp.hd AND ram = Comp.ram)
```

Рассмотрим пример, иллюстрирующий операцию разности двух множеств:

```
SELECT DISTINCT model, ram, hd
FROM PC AS Comp
WHERE ram < 1024 AND NOT EXISTS
(SELECT model FROM PC
WHERE hd < Comp.hd AND ram = Comp.ram)
```

На рисунке 5.18, а представлен запрос, который содержит информацию, о компьютерах с размером оперативной памяти менее 1024. На рисунке 5.18, б представлен результат пересечения исходного множества с множеством компьютеров, у которых размер оперативной памяти такой же, но существуют еще модели с меньшим размером жесткого диска. На рисунке 5.18, в представлен результат разности исходного множества с множеством компьютеров, у которых размер оперативной памяти такой же, но существуют еще модели с меньшим размером жесткого диска. Следует заметить, что операции пересечения и разности взаимодополняющие, если сложить результаты обоих запросов, то получится исходное множество.

model	hd	ram
1234	20	256
1456	40	512
1333	30	512
1782	10	256
1121	20	256

а)

model	hd	ram
1234	20	256
1456	40	512
1121	20	256

б)

model	hd	ram
1333	30	512
1782	10	256

в)

Рисунок 5.18 – Операция явного соединения таблиц:
а) исходное множество, б) операция пересечения, в) операция разности

Как можно заметить в предыдущих примерах применялся подзапрос. **Подзапросом** является выборка, результаты которой используются в предикате.

Язык структурированных запросов SQL предоставляет возможность решать одни и те же задачи несколькими различными способами. При этом наиболее компактными и производительными являются способы, которые используют подзапросы, вложенные в основной запрос.

В общем случае запрос возвращает множество значений. Поэтому использование подзапроса в предикате без операторов EXISTS, IN, ALL и ANY, которые возвращают булево значение, может привести к ошибке выполнения запроса. Использование оператора EXISTS уже было рассмотрено, изучим, как применяются операторы ALL и ANY для построения подзапросов.

Если для какого-нибудь значения, получаемого из подзапроса, результат операции равняется TRUE, то предикат ANY также равняется TRUE:

```
SELECT DISTINCT ram, model
FROM PC AS A
WHERE model = ANY
(SELECT model
FROM PC
WHERE ram < A.ram)
```

Предикат ALL исполняется так же, как и ANY, но для всех значений, получаемых из подзапроса, проверка должна удовлетворять результату TRUE:

```
SELECT DISTINCT ram, model
FROM PC AS A
WHERE model = ALL
(SELECT model
FROM PC
WHERE ram < A.ram)
```

Результат выполнения этих запросов представлен на рисунке 5.19.

ram	model
512	1456
512	1333
1024	2431

а)

ram	model
1024	2431

б)

Рисунок 5.19 – Применение подзапросов:
а) с использование оператора ANY, б) с использование оператора ALL

Использование параметров ANY и ALL требует от разработчиков запросов придерживаться определенных правил, остановимся на них более подробно:

1) если определен параметр ALL или ANY, и все результаты сравнения значения выражения и каждого значения, полученного из подзапроса, являются TRUE, истинностное значение равно TRUE;

2) если результат выполнения подзапроса не содержит строк, и определен параметр ALL, результат равен TRUE. Если же определен параметр ANY, результат равен FALSE;

3) если определен параметр ALL, и результат сравнения значения выражения хотя бы с одним значением, полученным из подзапроса, является FALSE, результирующее значение равно FALSE;

4) если определен параметр SOME, и хотя бы один результат сравнения значения выражения и значения, полученного из подзапроса, является TRUE, результирующее значение равно TRUE;

5) если определен параметр ANY и каждое сравнение значения выражения и значений, полученных из подзапроса, равно FALSE, результирующее значение тоже равно FALSE;

6) в любом другом случае результат будет равен UNKNOWN.

Расположение подзапросов в операторе SELECT может быть различным: в предложении WHERE, FROM и SELECT. Рассмотрим применение подзапроса в предложении WHERE:

```
SELECT model, price
FROM PC
WHERE price >
(SELECT AVG(price)
FROM PC)
```

Рассмотрим применение подзапроса в предложении FROM:

```
SELECT ram, Avg_price
FROM
(SELECT ram, AVG(price) AS Avg_price
FROM PC
GROUP BY ram)
WHERE pr > 700
```

Рассмотрим применение подзапроса в предложении SELECT:

```
SELECT 'разница' AS comment,
(SELECT AVG(price)
FROM PC
WHERE hd >= 40) –
(SELECT AVG(price)
FROM PC
WHERE hd < 40) AS price
```

Результаты описанных выше запросов приведены на рисунке 5.20.

model	price
1456	850
1333	700
2431	920

а)

ram	Avg_price
512	775
1024	920

б)

comment	price
разница	280

в)

Рисунок 5.20 – Применение подзапросов в предложении:
а) WHERE, б) FROM, в) SELECT

Использование оператора SELECT позволяет реализовать большинство реляционных операций, поэтому знание всех возможностей данного оператора позволит разрабатывать приложения максимальной производительности и любой сложности.

5.3 Оператор INSERT

Оператор INSERT вставляет новые строки в таблицу. При этом значения столбцов могут представлять собой литеральные константы либо являться результатом выполнения подзапроса. Синтаксис данного оператора имеет вид:

INSERT INTO <имя таблицы>[(<имя столбца>,...)]
{**VALUES** (< значение столбца>,…)}
| <выражение запроса>
| {**DEFAULT VALUES**}

Список столбцов в данном операторе не является обязательным. В том случае, если он отсутствует, список вставляемых значений должен быть полный:

INSERT INTO PC
VALUES (980, 900, 80, 1024, '52x', 3355)

Если задать список столбцов, то можно изменить определенный при создании таблицы порядок их следования:

INSERT INTO PC (model, price, cd, speed, hd, ram)
VALUES (3355, 980, '52x', 900, 80, 1024)

Эта конструкция громоздка, однако, она становится более предпочтительной, если столбцы имеют значения по умолчанию:

INSERT INTO PC (model, price, speed)
VALUES (3355, 980, 900)

При этом также можно не указывать список столбцов:

INSERT INTO PC
VALUES (3355, 980, **DEFAULT**, **DEFAULT**, 80, **DEFAULT**)

При необходимости можно вставить все значения по умолчанию:

INSERT INTO Product
DEFAULT VALUES

В операторе INSERT можно использовать подзапрос, например, вставить в таблицу PC строки из аналогичной по структуре таблицы Computer:

INSERT INTO PC
SELECT *
FROM Computer
WHERE type = 'PC'

Также можно задать порядок следования столбцов:

```
INSERT INTO PC (model, price, speed)
SELECT model, price, speed
FROM Computer
WHERE type = 'PC'
```

Преодолеть ограничение на вставку одной строки в операторе INSERT при использовании предложения VALUES позволяет применение подзапроса, формирующего строку с предложением UNION ALL:

```
INSERT INTO PC
SELECT 980, 900, 80, 1024, '52x', 3355
UNION ALL
SELECT 850, 780, 40, 1024, '50x', 3577
```

5.4 Оператор UPDATE

Оператор UPDATE изменяет имеющиеся данные в таблице. Команда имеет следующий синтаксис:

```
UPDATE <имя таблицы>
SET {имя столбца = {выражение
| NULL
| DEFAULT},...}
[{WHERE <предикат>}]
```

С помощью одного оператора могут быть заданы значения для любого количества столбцов. Однако, в одном операторе UPDATE можно вносить изменения в каждый столбец указанной таблицы только один раз.

Если столбец допускает NULL-значение, то его можно указать в явном виде. Кроме того, можно заменить исходное значение значением по умолчанию (DEFAULT) для данного поля. Для изменения значения какого-либо столбца можно использовать имена модифицируемых столбцов, например:

```
UPDATE PC
SET price=price / 2
```

Разрешается также значения одних столбцов присваивать другим:

```
UPDATE PC
SET hd = ram / 2
WHERE hd < 10
```

Если требуется изменять данные в зависимости от содержимого некоторого столбца, можно воспользоваться выражением CASE:

```
UPDATE PC
SET hd = CASE
WHEN ram < 128
THEN 20
ELSE 40 END
```

Для вычисления значений допускается использование подзапросов:

```
UPDATE PC
SET speed =
(SELECT MAX(speed)
FROM Computer)
```

5.5 Оператор DELETE

Оператор DELETE удаляет строки из таблиц и имеет следующий синтаксис:

```
DELETE FROM <имя таблицы >
[WHERE <предикат>]
```

В качестве предиката можно использоваться аналогичные конструкции, рассмотренные при изучении выборки данных, например:

```
DELETE FROM PC
WHERE ram <= 256
```

Если необходимо очистить таблицу от данных, то нужно использовать следующую конструкцию:

```
DELETE FROM PC
```

При проведении операции удаления данных из таблиц также можно использовать подзапросы:

DELETE FROM PC
WHERE model **NOT IN**
(**SELECT** model **FROM** Computer)

5.6 Функции SQL

Функции представляют собой подпрограммы, возвращающие значения, иногда основываясь на значениях входных параметров. Функции можно использовать везде, где разрешены выражения, в том числе вместо параметров самих функций, то есть допускаются вложенные вызовы функций.

Функция принимает параметры в круглых скобках, между именем функции и открывающей скобкой не должно быть пробелов.

В большинстве случаев применяются три типа функций: математические, строковые функции и функции даты/времени (рисунок 5.21).

Рисунок 5.21 – Классификация функций SQL

Рассмотрим кратко основные группы функций SQL и их использование в запросах, подробно не останавливаясь на их синтаксисе.

5.6.1 Математические функции

Математические функции выполняют различные математические операции. В качестве аргументов большинство из них принимает числа с плавающей запятой и возвращает результат аналогичного типа. Кратко рассмотрим наиболее используемые математические функции (таблица 5.1).

Таблица 5.1 – Список математических функций

Функция	Описание
ABS(<число>)	возвращает модуль числа
COS(<число>)	возвращает косинус числа в радианах
SIN(<число>)	возвращает синус числа
TAN(<число>)	возвращает тангенс числа

Продолжение таблицы 5.1

LOG(<число>)	возвращает натуральный логарифм числа
LOG10(<число>)	возвращает десятичный логарифм числа
RADIANS(<число>)	переводит градусы в радианы
EXP( )	основание натурального логарифма
PI( )	число π
POWER(<число>,<число>)	возводит число в степень
SQRT(<число>)	извлекает квадратный корень
MOD(<число>,<число>)	возвращает остаток от деления первого числа на второе
RAND( )	возвращает случайное число в интервале от 0 до 1
ROUND(<число>,<число>)	округляет число с плавающей запятой
SEILING(<число>)	округляет число до ближайшего большего целого числа
FLOOR(<число>)	округляет до ближайшего меньшего целого числа
SIGN(<число>)	возвращает –1, если число отрицательное, и +1, если число положительное

Рассмотрим применение некоторых из этих функций.

Пример на использование тригонометрической функции и функции округления показывает способ вычисления площади по данным таблицы figure:

```
SELECT ROUND(a*b*COS(angel)/2,2) AS square
FROM figure
WHERE type = 'triangle'
```

square
23.04
12.98
55.55
40.00

а)

volume
123.33
368.12
777.07
211.11

б)

Рисунок 5.22 – Результаты выполнения математических функций

Аналогично можно использовать константы:

```
SELECT ROUND(height*PI( )*POWER(Radius,2),2) AS volume
FROM figure
WHERE type = 'cylinder'
```

Результаты запросов представлены на рисунке 5.22.

5.6.2 Строковые функции

Строковые функции принимают строки в качестве аргументов и возвращают строки. Затем полученный с помощью применения этих функций результат можно использовать для построения предикатов, например, с помощью оператора LIKE. Рассмотрим наиболее используемые строковые функции (таблица 5.2).

Таблица 5.2 – Список строковых функций

Функция	Описание
CHAR_LENGTH(<строка>)	возвращает количество символов в строке
LCASE(<строка>)	символы строки переводятся в нижний регистр
RCASE(<строка>)	символы строки переводятся в верхний регистр
LEFT(<строка>, <число>)	извлекает из строки подстроку заданной длины слева
RIGHT(<строка>, <число>)	извлекает из строки подстроку заданной длины справа.
LTRIM(<строка>)	удаляет из строки начальные пробелы слева
RTRIM(<строка>)	удаляет из строки начальные пробелы справа
SPACE(<число>)	возвращает строку, состоящую из пробелов
REVERSE(<строка>)	меняет порядок символов в строке
CONCAT(<строка>,…)	объединяет группу строк
INSERT(<строка>, <число>, <число>, <строка>)	вставляет в строку заданную подстроку
LOCATE(<подстрока>, <строка>, <число>)	возвращает позицию первого вхождения подстроки
REPEAT(<строка>, <число>)	строка состоит из повторяющихся строк
REPLACE(<строка>, <подстрока>, <подстрока>)	меняет вхождение одной подстроки на новую подстроку

Рассмотрим пример использования строковых функций. Извлечение из названия геометрических фигур таблицы figure первой буквы слева происходит следующим образом:

```
SELECT DISTINCT LEFT(type, 1) AS letter
FROM figure
ORDER BY 1
```

Пример замены вхождения одной подстроки на новую подстроку выглядит так:

```
SELECT REPLACE(type, 'c', 's')
FROM figure
WHERE LOCATE('c', type) > 0
```

Функция определения длины строки применяется следующим образом:

```
SELECT name
FROM ships
WHERE CHAR_LENTH(RTRIM(name))>7
```

Результаты вышеприведенных запросов представлены на рисунке 5.23.

letter
c
s
t

а)

letter
sylinder
sube

б)

letter
cylinder
triangle

в)

Рисунок 5.23 – Результаты выполнения строковых функций

5.6.3 Функции работы с датой и временем

Функции работы с датой и временем используют значения даты/времени. При извлечении значений из базы данных, они приводятся к числовому или строковому типу, в зависимости от контекста. Большое число задач требует применения группировки данных для того, чтобы получить итоги по интервалам времени. Решить такие задачи можно с помощью применения функций даты/времени. Рассмотрим наиболее используемые из этих функций (таблица 5.3).

Таблица 5.3 – Список функций работы с датой и временем

Функция	Описание
NOW( )	функция определяет текущую дату и время
HOUR(<время>)	номер часа для заданного времени
MINUTE(<время>)	номер минуты для заданного времени
DAYNAME(<дата>)	возвращает название дня для заданной даты
DAYOFWEEK(<дата>)	возвращает номер дня недели для заданной даты
DAYOFMONTH(<дата>)	возвращает номер дня месяца для заданной даты
DAYOFYEAR(<дата>)	возвращает номер дня с начала года
WEEK(<дата>)	функция возвращает номер недели для заданной даты
MONTH(<дата>)	возвращает номер месяца для заданной даты
YEAR(<дата>)	возвращает номер года для заданной даты
DATE_ADD(<дата>, <интервал>)	прибавляет к значению даты заданный интервал
DATE_SUB(<дата>, <интервал>)	вычитает из значения даты заданный интервал
PERIOD_ADD(<дата>, <число>)	прибавляет число месяцев к заданному периоду
PERIOD_DIFF(<дата>, <число>)	определяет разницу в месяцах между периодами

Рассмотрим пример использования функций работы с датой и временем на таблице продаж товаров Sale. Таблица содержит столбец date, в котором хранится дата продажи, столбец sum, в котором храниться общая стоимость операции.

Следующий пример иллюстрирует использование функции вычитания из текущей даты заданного интервала в днях:

```
SELECT DATE_SUB(NOW( ) INTERVAL DAYOFYEAR
(SELECT MAX(date)
FROM sale)
DAY) AS day
```

Данный пример показывает, как получать итоговую информацию за неделю по дням, используя функцию получения номера недели для заданной даты и функцию определения название дня для заданной даты:

```
SELECT DAYNAME(date) AS day, SUM(sum) AS total
FROM sale WHERE WEEK(NOW()) = WEEK (date)
GROUP BY DAYNAME(date)
```

В следующем примере показана группировка количества операций по дням текущего месяца, выполненная с помощью функций, которые определяют номер дня месяца и номер месяца для заданной даты:

```
SELECT DAYOFMONTH(date) AS day,
COUNT(sum) AS quantity FROM sale
WHERE MONTH(NOW()) = MONTH(date)
GROUP BY DAYOFMONTH(date)
```

Результаты вышеприведенных запросов представлены на рисунке 5.24.

day
8

а)

day	total
Monday	22008
Thursday	19455
Wednesday	35972

б)

day	quantity
1	50
2	43
7	32
10	17
12	5
19	22
20	16

в)

Рисунок 5.24 – Результаты выполнения функций работы с датой и временем

Представленный выше список функции далеко не полон, однако он дает представление о возможностях SQL по обработке данных, и изучение приведенных примеров позволит строить запросы произвольной сложности для решения практических задач при проектировании баз данных и разработке приложений.

Тезаурус

Предикат, трафаретный символ, выборка, агрегирование, группировка, условный оператор, запрос, подзапрос, явное соединение таблиц, декартово произведение, добавление данных, обновление данных, удаление данных, функция, оператор, интервал, типы данных.

Контрольные вопросы

1) В чем отличие горизонтальной выборки от вертикальной?
2) Какие существуют трафаретные символы?
3) Какие значения может принимать предикат в SQL?
4) Что означает ключевое слово DISTINCT?
5) Назовите правила использования агрегатных функций в операторе SELECT?
6) В чем отличие предложений WHERE и HAVING?
7) Какова разница между левым и полным соединением таблиц?
8) Как вставить несколько записей с использованием одной команды INSERT?
9) Можно ли в SQL добавлять пустые записи?
10) Чем отличается функция REPEAT( ) и REVERSE( )?

Список рекомендуемой литературы

1) Андон, Ф. Язык запросов SQL [Текст] / Ф. Андон, В. Резниченко. – СПб. : BHV, 2006. – 416 с.
2) Аткинсон, Л. MySQL Библиотека профессионала [Текст] / Л. Аткинсон. – М. : Вильямс, 2002. – 624 с.
3) Дунаев, В. В. Базы данных. Язык SQL [Текст] / В. В. Дунаев. – СПб. : BHV, 2006. – 288 с.
4) Клайн, К. SQL. Справочник [Текст] / К. Клайн. – М. : КУДИЦ-ОБРАЗ, 2006. – 832 с.
5) Мартин, Г. SQL. Бестселлер#1. Описание SQL92, SQL99 и SQLJ [Текст] / Г. Мартин. – М. : Лори, 2004. – 644 с.
6) Моисеенко, С. И. SQL. Задачи и решения [Текст] / С. И. Моисеенко. – СПб. : Питер, 2006. – 256 с.
7) Тоу, Д. Настройка SQL. Для профессионалов [Текст] / Д. Тоу. – СПб. : Питер, 2004, – 333 с.

ГЛАВА 6
РАЗРАБОТКА БАЗ ДАННЫХ С ПОМОЩЬЮ MICROSOFT OFFICE ACCESS 2007

В настоящей главе рассматриваются вопросы проектирования баз данных, разработки запросов, создания отчетов и форм в Microsoft Office Access 2007.

6.1 Разработка таблиц в Microsoft Office Access 2007

При решении многих прикладных задач порой не требуется привлечения мощных средств разработки и группы профессиональных программистов. Например, если база данных должна хранить небольшой объем информации, интерфейс клиентских приложений должен быть интуитивно понятен и прост, то в качестве эффективной системы управления базами данных можно использовать Microsoft Office Access 2007, который является лидером среди подобного класса программных продуктов. Изучим этапы проектирования баз данных и клиентских приложений в этой среде.

Рассмотрим в качестве примера базу данных «Студенты», которая состоит из трех таблиц: «Студенты», «Дисциплина» и «Результат» (рисунок 6.1).

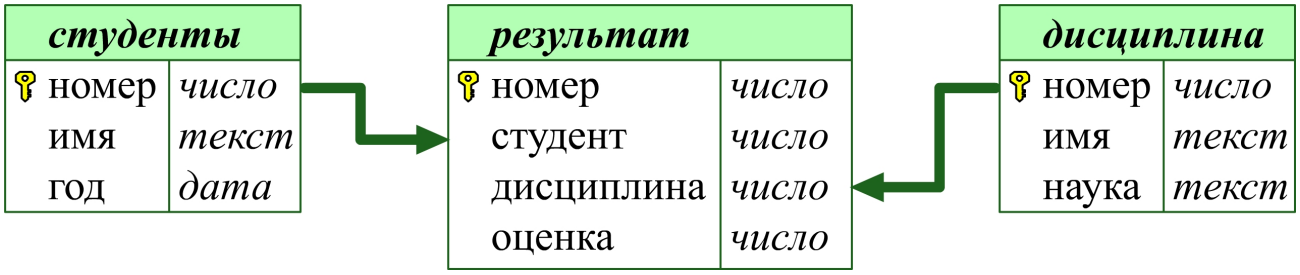


Рисунок 6.1 – Структура базы данных «Студенты»

Создать базу данных в Microsoft Office Access 2007 можно с помощью шаблонов, ориентированных на определенную предметную область. В большинстве случаев найти полностью удовлетворяющий целям проекта шаблон не удается, поэтому приходится разрабатывать базу данных с самого начала. Для создания базы данных необходимо на этапе запуска Microsoft Office Access 2007 выбрать пункт «Новая база данных» и указать ее имя, при этом первая таблица создается автоматически.

Для создания следующих таблиц необходимо перейти на закладку «Создать» ленты Microsoft Office и выбрать пункт «Таблица» в группе «Таблица» (рисунок 6.2).

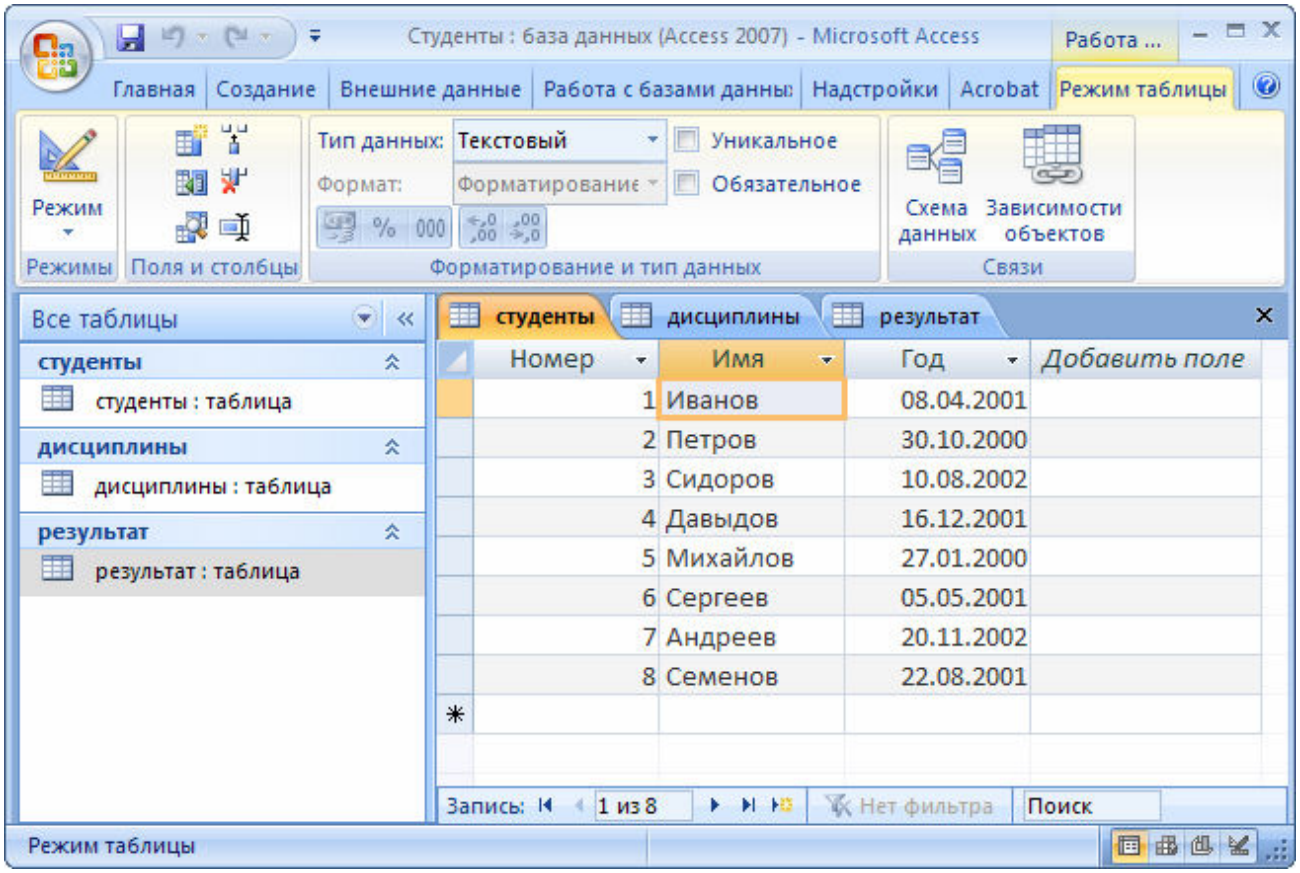


Рисунок 6.2 – Создание таблицы «Студенты»

Для каждой таблицы можно в режиме ввода данных или в режиме конструктора определить количество используемых столбцов, их тип, задать первичный ключ и индексы. В Microsoft Office Access 2007 определено восемь типов данных: текстовый, числовой, дата/время, денежный, логический, поле MEMO, гиперссылка, вложение. Практически все они стандартные и работа с этими типами данных не представляет трудностей. Наиболее интересным и новым является тип вложение, который позволяет хранить внешние данные, например, фотографии, звуковые файлы, видеоданные.

Для каждого столбца помимо типа можно определить формат хранения информации, а также установить контроль ввода уникальных и ненулевых значений. Именно по такому принципу разработаны таблицы «Студенты» и «Дисциплины».

Таблица «Результат» содержит два внешних ключа для связи с разработанными таблицами. Microsoft Office Access 2007 предоставляет эффективный инструмент связи таблиц между собой. Для этого необходимо на ленте Microsoft Office на закладке «Режим таблицы» в группе «Поля и столбцы» выбрать элемент «Столбец подстановок». После этого нужно указать, значения из какого столбца и таблицы будут подставляться в исходную таблицу. Такую операцию необходимо проделать для создания всех полей, содержащих внешние ключи (рисунок 6.3).

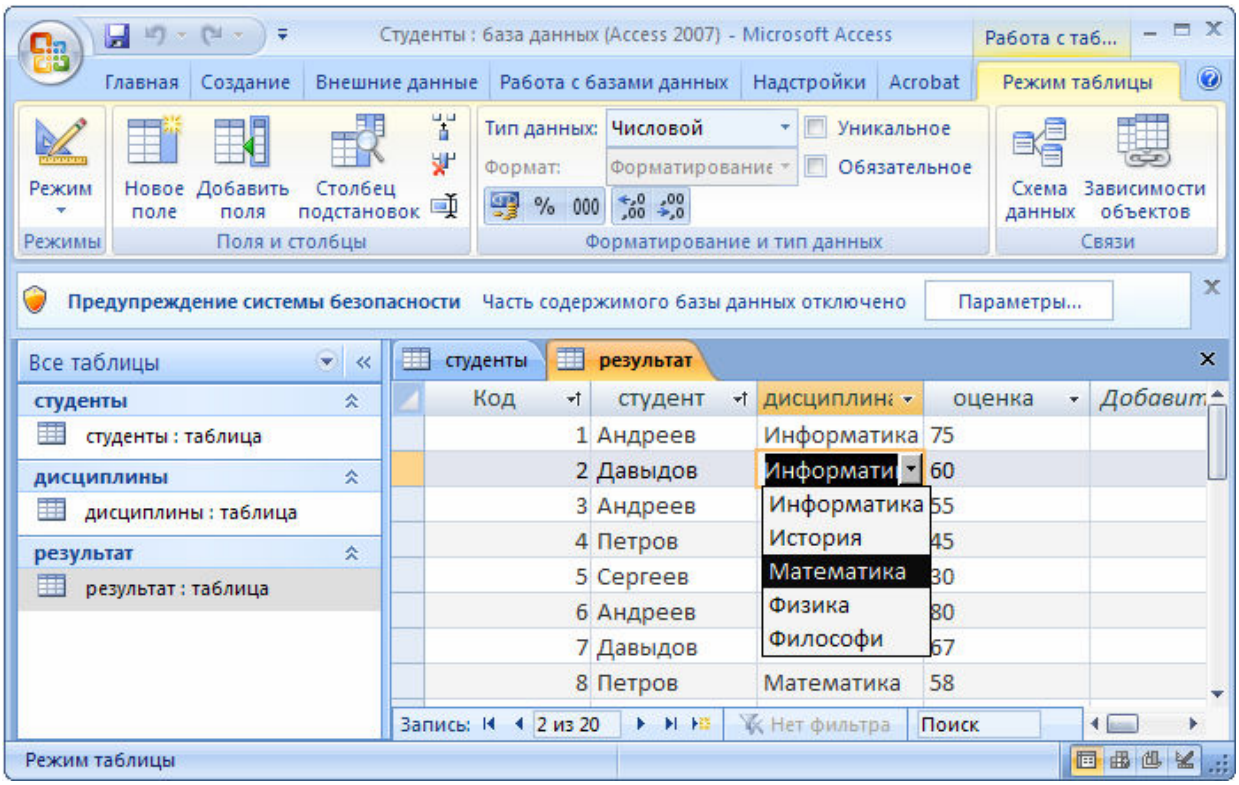


Рисунок 6.3 – Создание столбцов подстановок для таблицы «Результат»

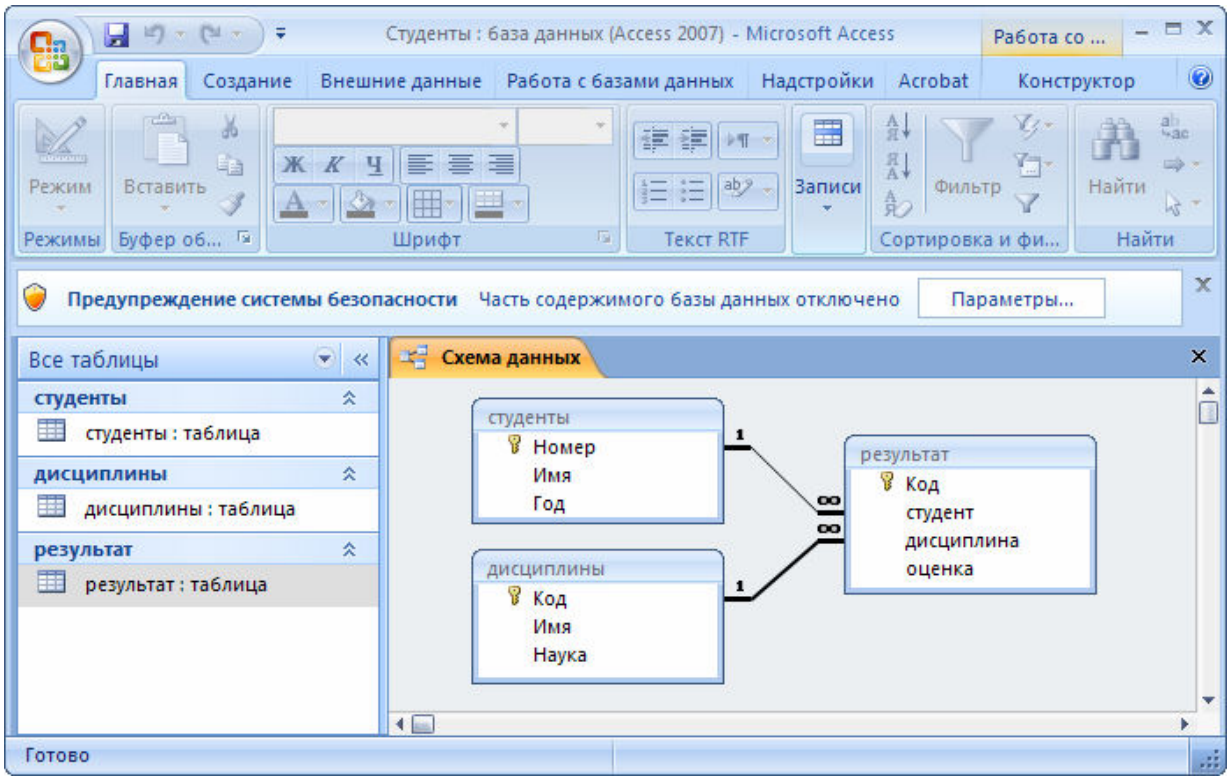


Рисунок 6.4 – Схема базы данных «Студенты»

Для того, чтобы посмотреть и отредактировать схему базы данных, нужно на ленте Microsoft Office на закладке «Режим таблицы» в группе «Связи» выбрать элемент «Схема данных» (рисунок 6.3), после чего представленную схему данных можно отредактировать, настроить связи между таблицами, убрать или добавить представления.

Все записи таблиц можно отсортировать по возрастанию или убыванию либо отфильтровать, причем сложность фильтра может быть различной. Для этого необходимо на ленте Microsoft Office на закладке «Главная» выбрать группу «Сортировка и фильтр». Более профессиональным способом фильтрации и сортировки осуществляются с помощью запросов, которые будут рассмотрены ниже.

6.2 Разработка форм в Microsoft Office Access 2007

Мощным средством визуализации информации, хранящейся в базе данных в Microsoft Office Access 2007, являются формы. Формы представляют собой интерфейсный элемент, который позволяет осуществлять взаимодействие между пользователем и базой данных. В формах пользователи осуществляют навигацию, редактирование и удаление данных.

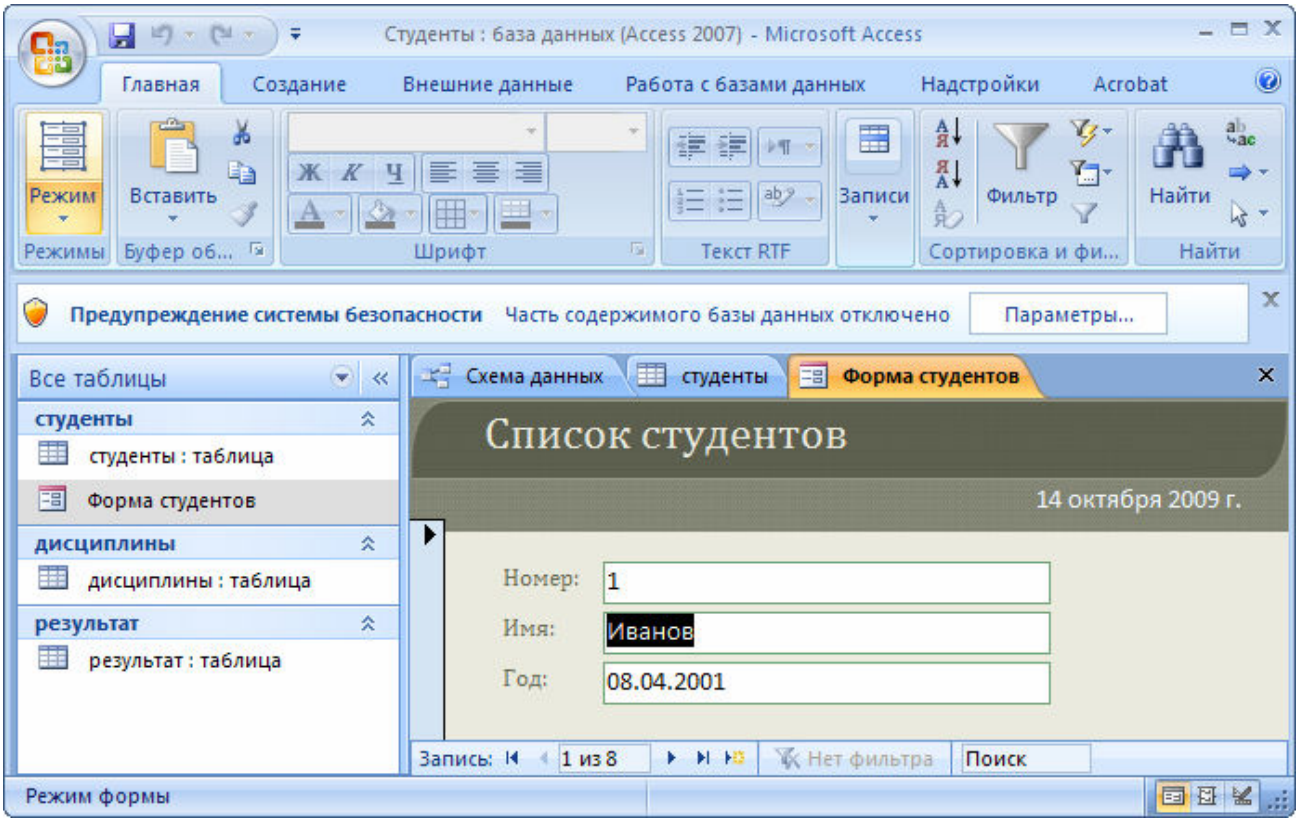


Рисунок 6.5 – Форма «Список студентов»

Рассмотрим способы создания форм в Microsoft Office Access 2007. Для этого на ленте Microsoft Office на закладке «Создать» в группе «Форма» необходимо выбрать пункт «Пустая форма». После этого нужно из списка полей переместить столбцы, которые будут отражены на данной форме. Данный способ создания форм осуществляется в режиме формы. На форме можно просматривать и редактировать каждую строку, переходить от одной строки к другой, редактировать, осуществлять поиск и фильтрацию информации.

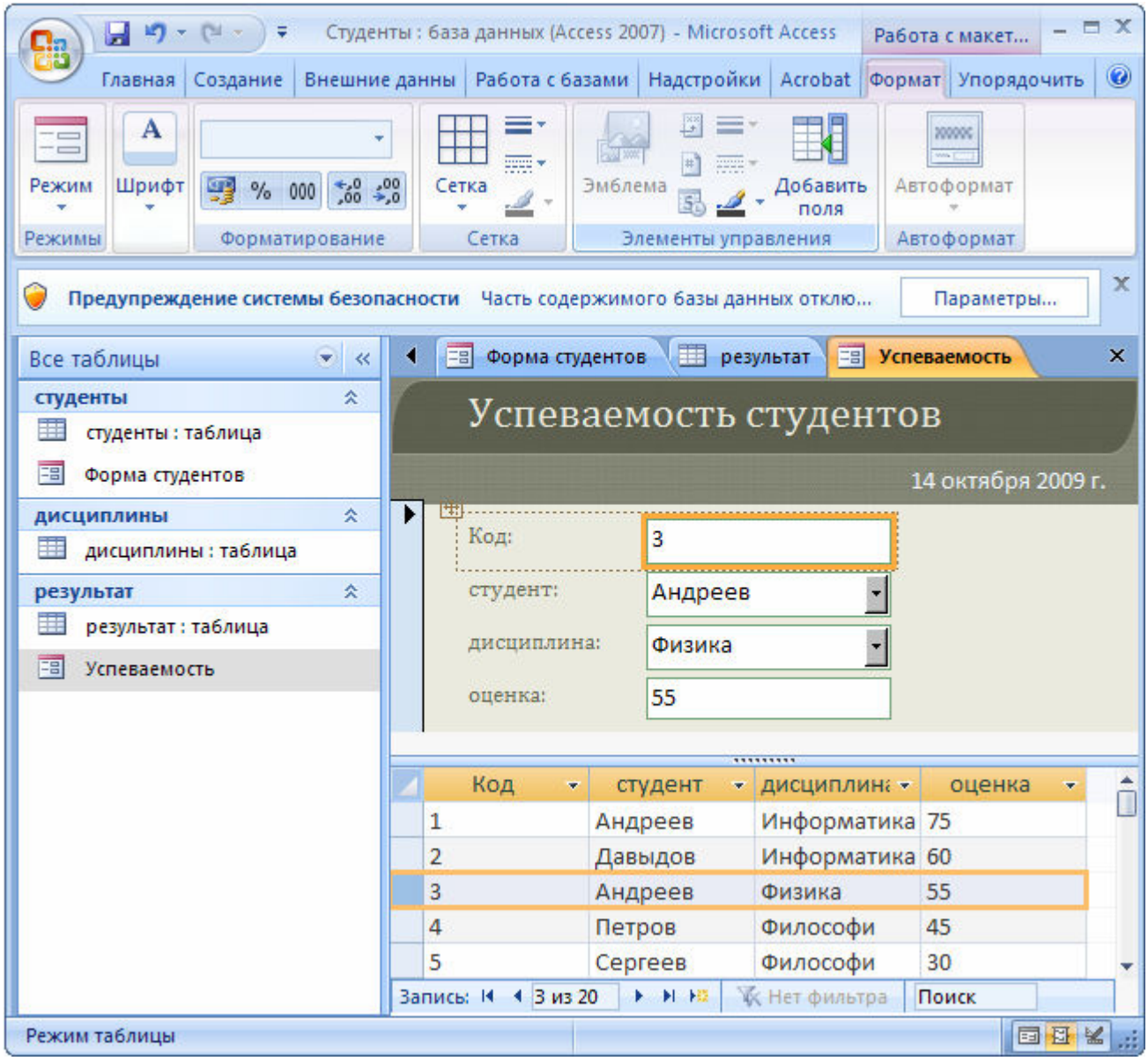


Рисунок 6.6 – Разделенная форма «Успеваемость студентов»

При необходимости можно использовать режим конструктора, который позволяет более детально проработать форму и настроить все элементы управления,

или мастер создания форм, который предоставляет возможность быстрой разработки макета формы.

В некоторых случаях, когда полей в таблице большое количество, бывает удобно осуществлять навигацию по таблице, где представлены основные поля, а полную информацию о текущей записи выводить на форму. Такую возможность предоставляет Microsoft Office Access 2007 путем создания разделенных форм. Разработаем разделенную форму для таблицы «Результат». Для этого на ленте Microsoft Office на закладке «Создать» в группе «Форма» необходимо выбрать пункт «Разделенная форма» (рисунок 6.6).

6.3 Разработка отчетов в Microsoft Office Access 2007

Разработка форм и сложных запросов к базе данных направлена лишь на то, чтобы в удобном виде представить информацию пользователю, однако, на сегодняшний день ее необходимо еще и сохранить, причем оформление информации также значимо, как и сама информация. Для этих целей в Microsoft Office Access 2007 используются отчеты, которые обеспечивают вывод информации, хранящейся в базе данных, на экран монитора, принтер или внешний носитель информации. Причем существующие мощные средства разработки отчетов позволяют создавать дизайн практически любой сложности.

Microsoft Office Access 2007 предоставляет возможность создавать макет отчета для выбранной таблицы, использовать мастер отчетов либо разрабатывать отчет с самого начала. Последний способ для опытных разработчиков более приемлем, так как позволяет избежать избыточности и сложных конструкции при проектировании отчета.

Для создания отчета необходимо на ленте Microsoft Office на закладке «Создать» в группе «Отчеты» выбрать пункт «Пустой отчет». После этого в панели «Список полей» нужно выбрать соответствующие столбцы таблиц. Далее необходимо сгруппировать данные для получения среза информации, для этого нужно на ленте Microsoft Office на закладке «Формат» в группе «Группировка и итоги» выбрать пункт «Группировка», после чего выбрать столбец, по которому осуществляется группировка, и столбец, по которому осуществляется сортировка. В нашем случае сгруппируем данные в таблице «Результат» по дисциплинам и отсортируем данные по оценкам.

В каждой группе подведем итоги: выведем средний результат по дисциплине, для этого в свойствах группировки (нижняя панель «Группировка, сортировка и итоги») необходимо выбрать нужную группировку, далее – пункт «С итогами», установить итог на поле «Оценка», тип итогов – «Среднее», причем вывод осуществлять в примечании группы. Для пояснения итога группировки можно в контекстном меню для ячейки с результатом группировки выбрать пункт «Задать подпись».

После этого можно в отчет вставить заголовок, служебную информацию, например, дату или количество страниц, выбрать стиль, отформатировать при необходимости текстовую и числовую информацию и использовать в работе данный отчет (рисунок 6.7).

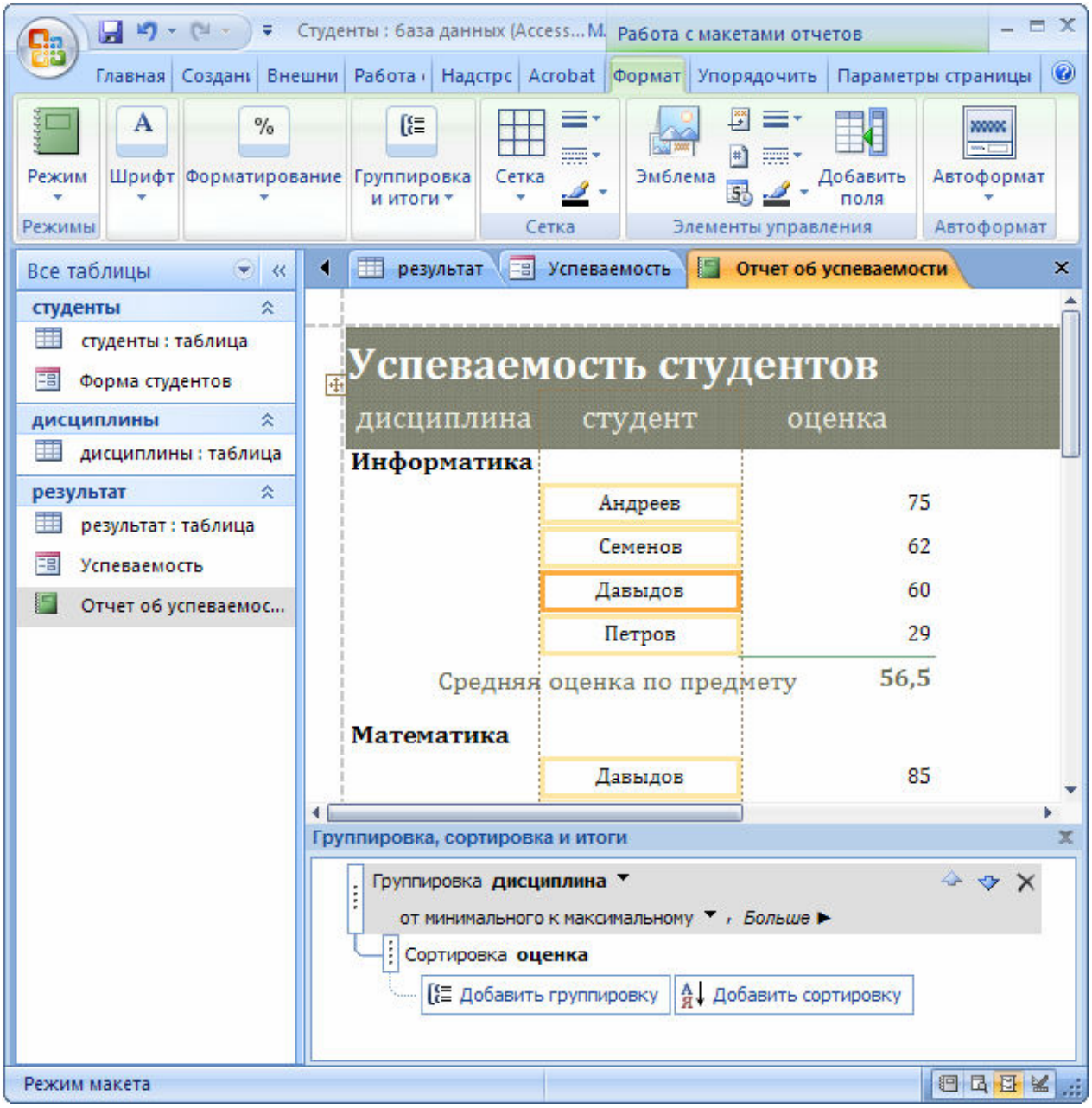


Рисунок 6.7 – Отчет «Успеваемость студентов»

Кроме вывода информации на экран монитора и принтер, данные можно экспортировать в различные форматы, например, в Microsoft Office Excel и в список Microsoft Office SharePoint.

6.4 Разработка запросов в Microsoft Office Access 2007

Запросы помогают сформировать выборки данных для представления их пользователю. В Microsoft Office Access 2007 существует отдельный инструмент, который реализует структурированный язык запросов SQL. Создавать запросы можно также несколькими способами: с помощью мастера запросов и с помощью конструктора. Разработаем запрос с помощью конструктора с самого начала.

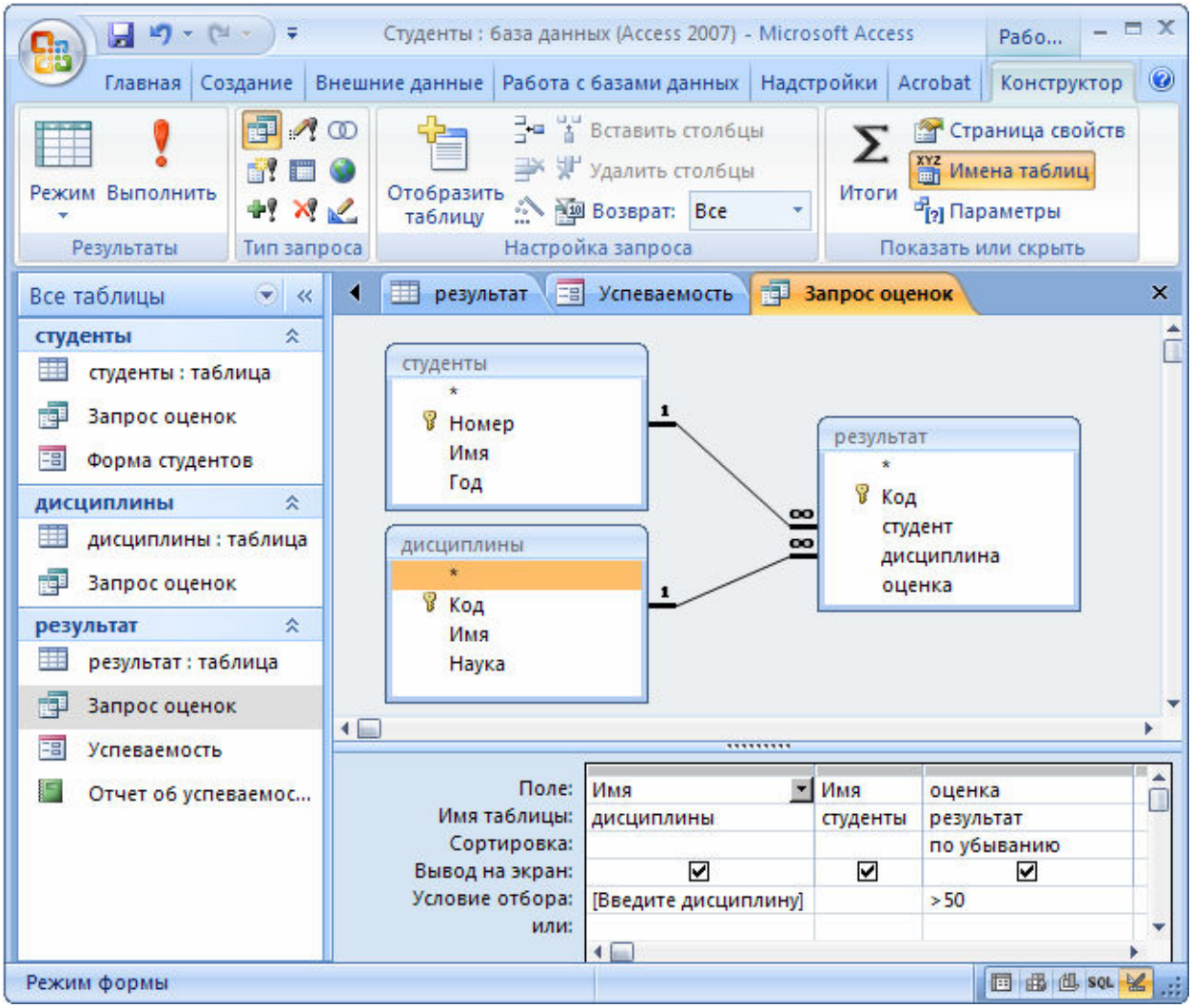


Рисунок 6.8 – Конструирование запроса «Оценки»

Для этого необходимо на ленте Microsoft Office на закладке «Создать» в группе «Другие» выбрать пункт «Конструктор запросов». Автоматически система предложит выбрать таблицы, из которых будут выбраны данные. Если в схеме данных до этого были установлены связи между таблицами, то они сохранятся, при этом их можно отредактировать и настроить под решаемую задачу.

После этого необходимо выбрать столбцы из таблиц, которые будут включены в выходной запрос. Далее по необходимости устанавливается порядок сортировки. В нашем примере установлена сортировка выходных данных по столбцу «Оценки» по убыванию (рисунок 6.8).

Затем определяется условие отбора данных с помощью булевых функций. Интересным представляется случай ввода параметра сравнения, при этом вместо условия устанавливается текст приглашения, заключенный в квадратные скобки. При запуске запроса первоначально необходимо ввести параметр, при совпадении значения в заданном столбце таблицы с введенным параметром строка включается в результат запроса. Например, выполнение разработанного запроса с параметром «Информатика» выдаст результат, показанный на рисунке 6.9.

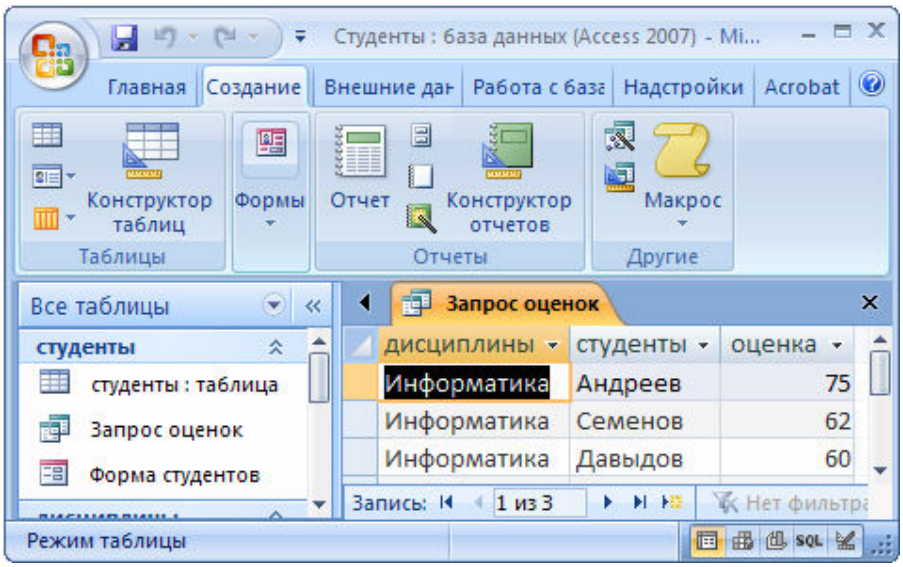


Рисунок 6.9 – Результат выполнения запроса «Оценки»

Автоматически после создания запроса формируется SQL инструкция, по мере необходимости ее можно исправлять или дорабатывать (листинг 6.1).

Листинг 6.1 – Код запроса «Оценки»

```
SELECT дисциплины.Имя, студенты.Имя, результат.оценка
FROM студенты
INNER JOIN (дисциплины
INNER JOIN результат
ON дисциплины.Код = результат.дисциплина)
ON студенты.Номер = результат.студент
WHERE (((дисциплины.Имя)=[Введите дисциплину])
AND ((результат.оценка)>50))
ORDER BY результат.оценка DESC;
```

Код данной инструкции полностью соответствует стандарту SQL2, хотя имеются и специфические особенности. Например, поддерживаются названия таблиц и столбцов на русском языке, что, конечно, повышает качество восприятия запроса, а также поддерживаются конструкции ввода параметров фильтрации из формы.

6.5 Использование диаграмм в Microsoft Office Access 2007

Для графического представления данных в Microsoft Office Access 2007 используются диаграммы. Именно использование диаграмм является дополнительным преимуществом Microsoft Office Access 2007, так как графическая интерпретация данных повышает наглядность и эффективность их восприятия.

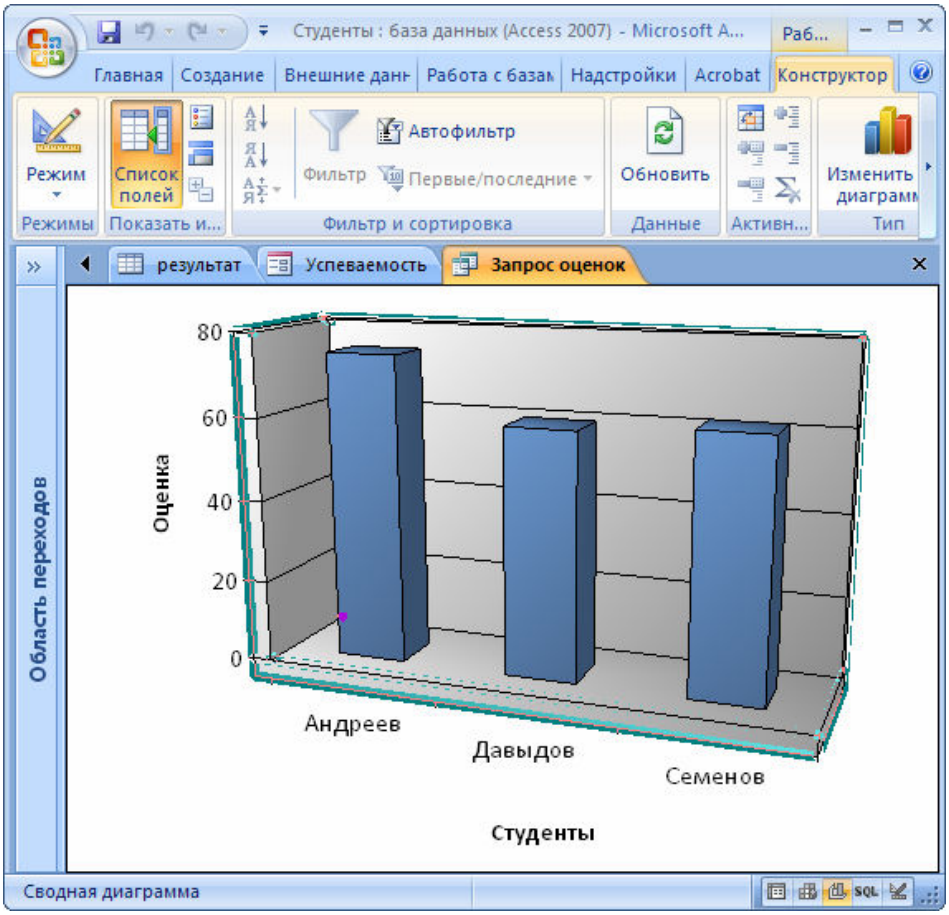


Рисунок 6.10 – Результат выполнения запроса «Оценки» в виде диаграммы

Диаграммы можно использовать в формах, таблицах, запросах. Рассмотрим пример создания диаграмм на основе разработанного запроса «Оценки».

Для вставки в запрос диаграммы необходимо на ленте Microsoft Office на закладке «Главная» в группе «Режим» выбрать пункт «Сводная диаграмма».

После этого нужно определить тип диаграммы, значения и категории, для чего из «Списка полей диаграммы» переместить необходимые столбцы в соответствующие области. Далее подписать названия осей, отформатировать с помощью стандартных средств текстовую и числовую информацию (рисунок 6.10). Когда отчет создан, можно, выбрав режим «Сводная диаграмма», просмотреть графическую интерпретацию результата запроса «Оценки».

Тезаурус

Шаблон, макет, конструктор, вложение, подстановка, форма, отчет, запрос, диаграмма.

Контрольные вопросы

1) Назовите существующие способы создания таблиц в Microsoft Office Access 2007.
2) Перечислите типы данных, используемых в Access 2007.
3) Для чего необходимы столбцы подстановок?
4) Как в Access 2007 можно отредактировать связи между таблицами?
5) В каких случаях используется разделенная форма?
6) Как в отчете Microsoft Office Access 2007 осуществляется расчет итогов по группам данных?
7) Какие типы диаграмм поддерживает Microsoft Office Access 2007?
8) Как установить первичный ключ для таблицы в Access 2007?
9) Для чего в Access 2007 используется режим конструктора?
10) Зачем в Access 2007 применяется условное форматирование?

Список рекомендуемой литературы

1) Сергеев, А. Access 2007. Новые возможности [Текст] / А. Сергеев. – СПб. : Питер, 2008. – 176 с.
2) Епанешников, А. М. Практика создания приложений в Access 2007 [Текст] / А. М. Епанешников, В. А. Епанешников. – М. : Диалог-МИФИ, 2009. – 440 с.
3) Тимошок, Т. В. Microsoft Office Access 2007 [Текст] / Т. В. Тимошок. – М. : Вильямс, 2008. – 464 с.
4) Рудикова, Л. В. Microsoft Office Access 2007 [Текст] / Л. В. Рудикова, И. А. Харитонова. – СПб. : BHV, 2008. – 1280 с.
5) Бекаревич, Ю. Б. Самоучитель Access 2007 [Текст] / Ю. Б. Бекаревич, Н. В. Пушкина. – СПб. : BHV, 2007. – 720 с.
6) Моисеенко, С. И. Разработка приложений в MS Access [Текст] / С. И. Моисеенко, Б. В. Соболь. – М. : Вильямс, 2006. – 272 с.
7) Кошелев, В. Е. Access 2007 [Текст] / В. Е. Кошелев. – М. : Бином, 2008. – 592 с.

ГЛАВА 7
РАЗРАБОТКА БАЗ ДАННЫХ С ПОМОЩЬЮ ТЕХНОЛОГИИ ADO.NET

В настоящей главе рассматриваются вопросы создания баз данных и клиентских приложение в Microsoft Visual Studio с помощью технологии ADO.NET.

7.1 Характеристика технологии ADO.NET

Приложения ADO.NET отличаются гибкостью и обширной поддержкой XML. Так как XML является форматом передачи наборов данных через сеть, обрабатывать данные может любой компонент, поддерживающий чтение данных в формате XML. Разработчики программного обеспечения, использующие данную технологию, могут разграничить обработку бизнес-логики приложения на сервере и обработку пользовательского интерфейса, выделив их в разные уровни, выполняемые на разных машинах, что позволяет упростить модификацию приложения. Компоненты данных ADO.NET в Visual Studio инкапсулируют функциональные возможности доступа к данным различными способами, что помогает разрабатывать программные продукты значительно быстрее и с меньшим количеством ошибок. ADO.NET способствует повышению масштабируемости при помощи стимулирования сбережения ограниченных ресурсов программистами. Так как любое приложение ADO.NET использует неподключенный доступ к данным, блокировки баз данных и подключения действуют кратковременно.

7.2 Создание базы данных для соединения с приложением

Создадим Windows-приложение, которое будет взаимодействовать с базой данных на языке программирования C#. Для этого необходимо запустить Visual Studio и с помощью команды «File\New Project» открыть новый проект – Windows-приложение, как показано на рисунке 7.1.

В Windows-приложении необходимо реализовать взаимодействие с базой данных. В качестве учебной базы данных рассмотрим базу данных «Гостиница», структура которой представлена на рисунке 7.2. Эта база данных содержит две таблицы. Таблица clients состоит из трех столбцов: столбец id_client содержит числовую информацию с кодом клиента, столбец name содержит имя клиента в текстовом виде, столбец address хранит адрес клиента в текстовом виде. Таблица order также состоит из трех столбцов: столбец id_order содержит числовую информацию с кодом заказа, столбец description хранит описание заказа в текстовом виде, столбец id_client является внешним ключом и содержит код клиента.

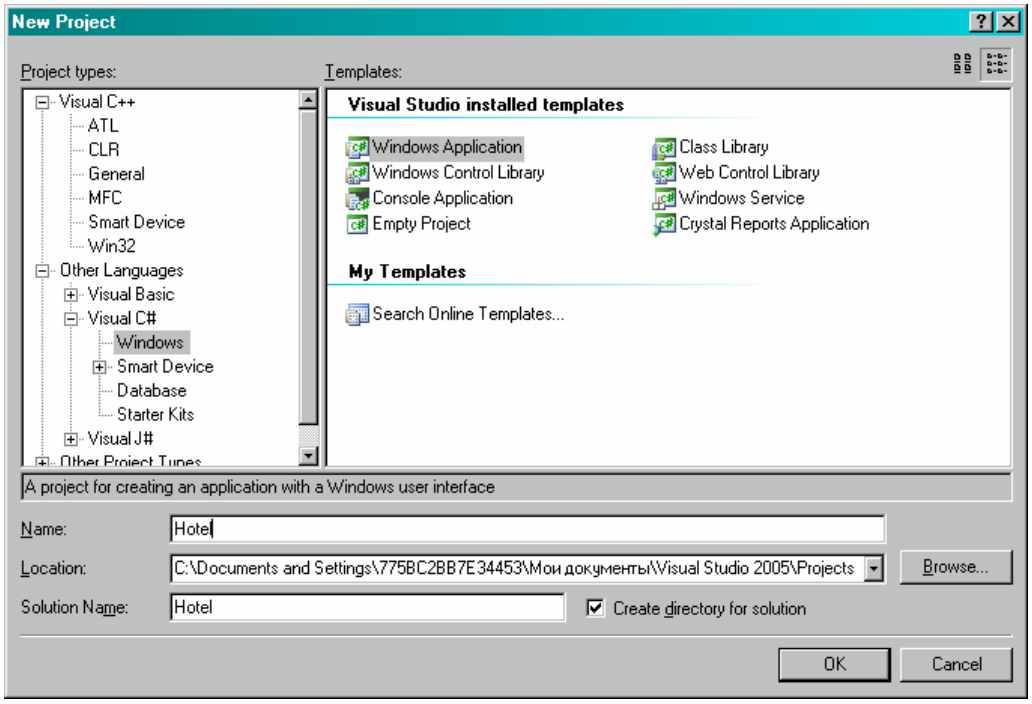


Рисунок 7.1 – Окно создания нового Windows-приложения

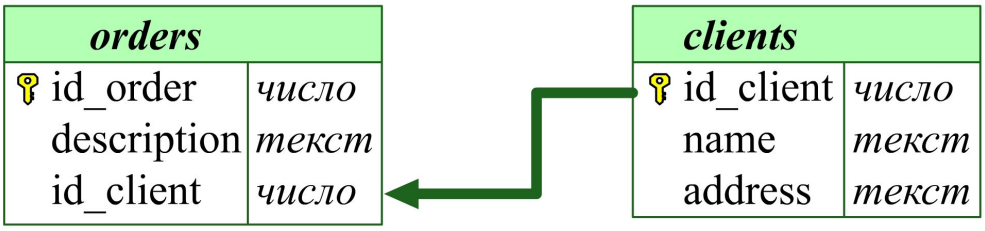


Рисунок 7.2 – Структура базы данных «Гостиница»

Для создания базы данных необходимо в главном меню выбрать «View\Server Explorer». Далее нужно в окне «Data Connections» выбрать пункт «Add Connection». В появившемся окне необходимо указать имя сервера «.\SQLExpress» и имя базы данных – hotel. Установка соединения представлена на рисунке 7.3.

Таким образом, создалась пустая база данных, в которую необходимо добавить таблицы и установить соответствующие связи между ними. Для этого нужно в контекстном меню «Tables» выбрать пункт «Add New Table». Добавим таблицу «order» в созданную базу данных. Заполним все столбцы, которые содержатся в данной таблице. id_order – первичный ключ данной таблицы, поэтому в контекстном меню данного поля необходимо выбрать «Set Primary Key» (рисунок 7.4).

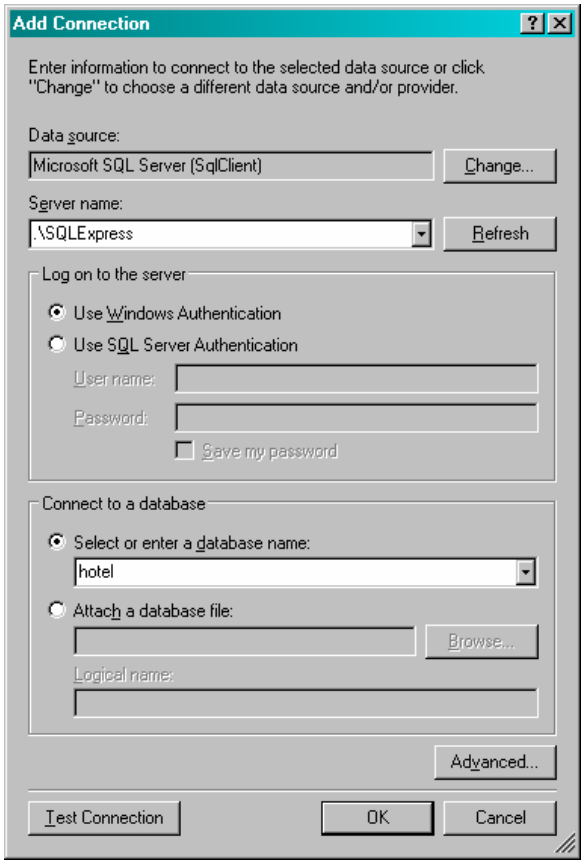


Рисунок 7.3 – Окно соединения с базой данных

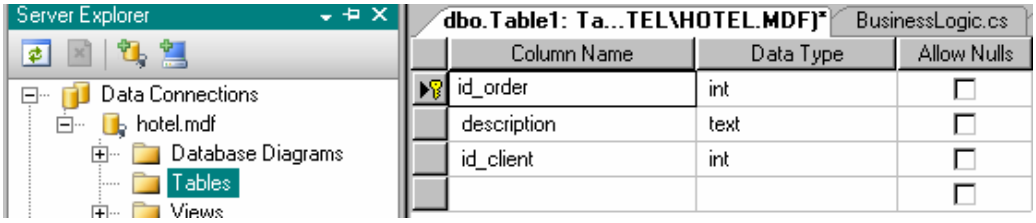


Рисунок 7.4 – Создание таблицы order

После закрытия данной вкладки необходимо ввести имя таблицы, которая добавится в папку «Tables». Аналогичным образом нужно добавить таблицу clients. Помимо таблиц orders и clients, входящих в логическую модель базы данных, добавим еще одну – generator. Таблица имеет единственный целочисленный столбец id, который является первичным ключом.

После создания всех таблиц установим связи между ними. В контекстном меню «Database Diagrams» выберем пункт «Add New Diagram». В появившемся окне нужно выбрать все имеющиеся таблицы и добавить их в диаграмму нажатием кнопки «Add» (рисунок 7.5).

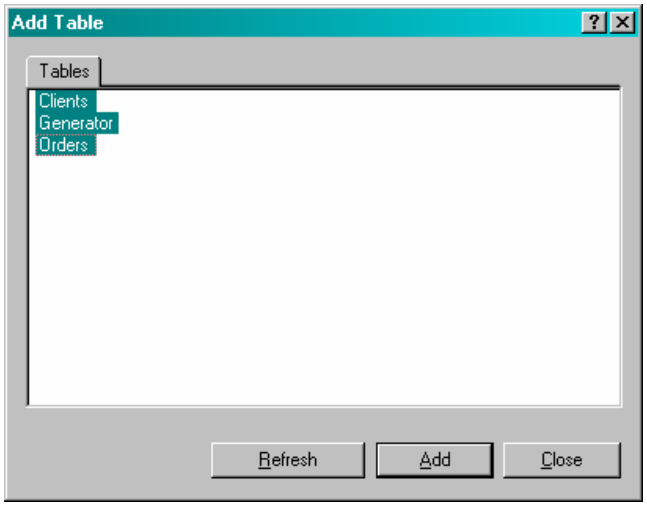


Рисунок 7.5 – Добавление таблиц в диаграмму

Далее необходимо установить связи на диаграмме. Для этого нужно выбрать в контекстном меню таблицы orders пункт «Relationships». Затем необходимо нажать кнопку «Add» и в окне «General Properties» выбрать пункт «Tables and Columns Specification», в котором выбрать таблицу с первичным ключом – clients и таблицу с внешним ключом – orders (рисунок 7.6).

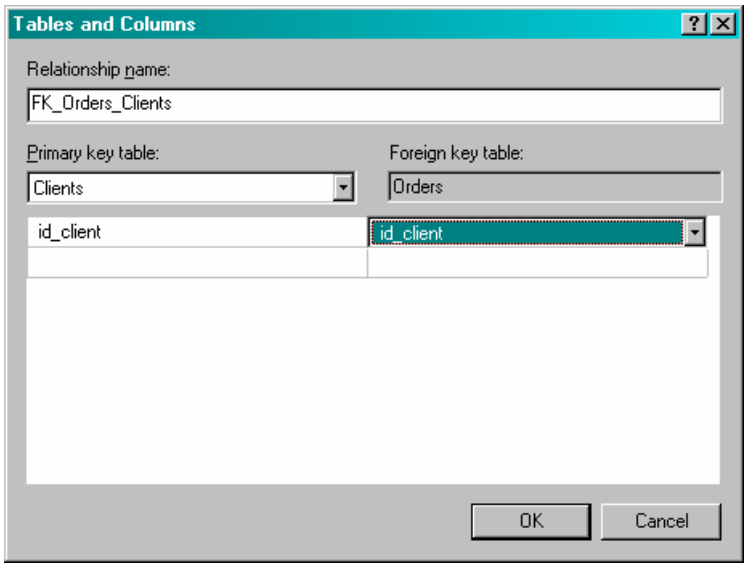


Рисунок 7.6 – Установка связи между таблицами orders и clients

Таким образом, установлена связь между таблицами orders и clients. Созданная диаграмма базы данных представлена на рисунке 7.7.

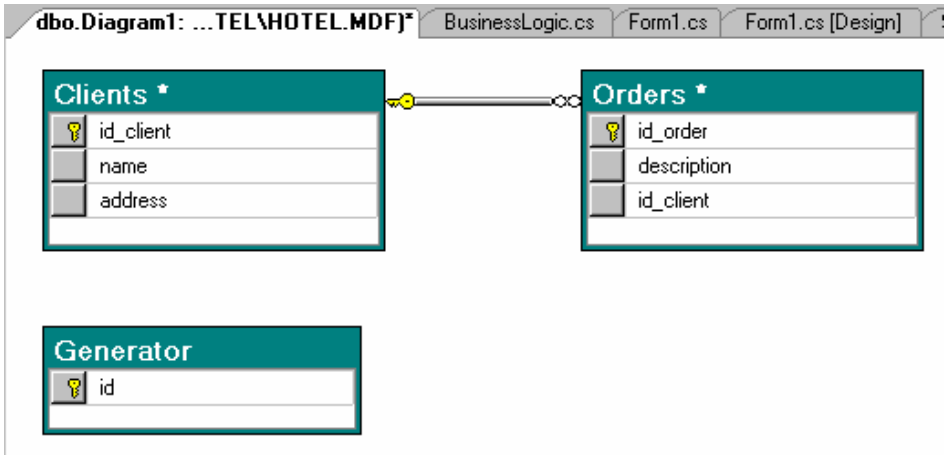


Рисунок 7.7 – Диаграмма базы данных

Теперь необходимо выбрать созданную базу как источник данных. Для этого в главном меню выберем «Data\Add New Data Source». В появившемся окне, укажем тип источника данных «Database» (рисунок 7.8).

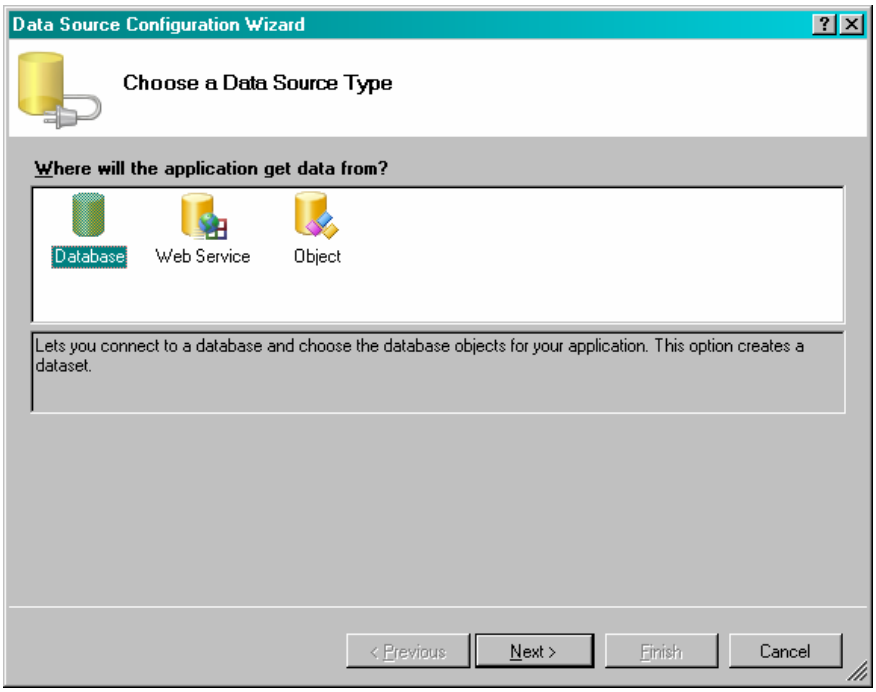


Рисунок 7.8 – Выбор источника данных

Далее необходимо выбрать созданное ранее соединение с базой данных (рисунок 7.9).

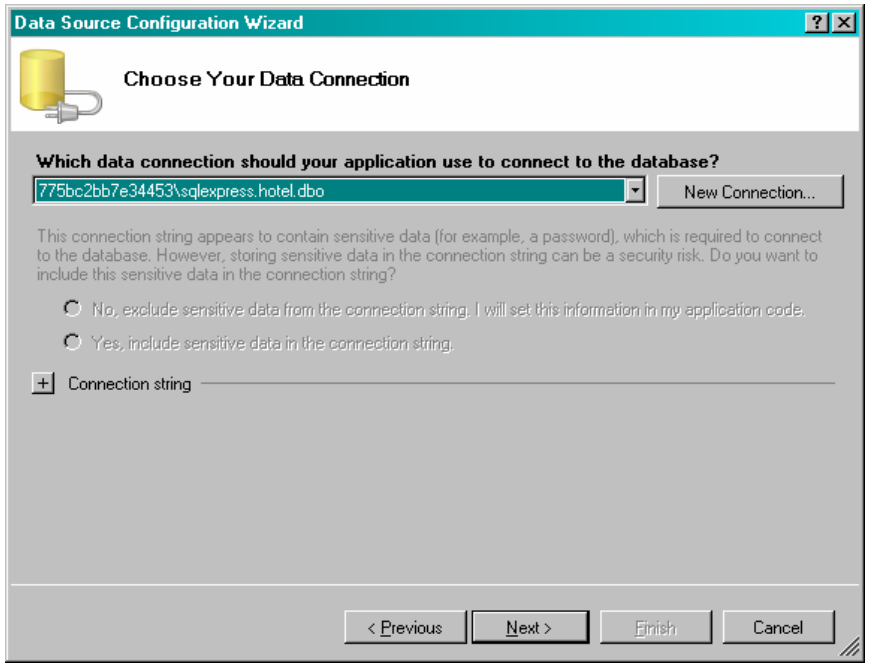


Рисунок 7.9 – Выбор соединения с базой данных

В следующем окне после выбора соединения необходимо сохранить строку подключения – ConnectionString. Далее нужно выбрать структуры базы данных, которые необходимо включить в новый источник данных.

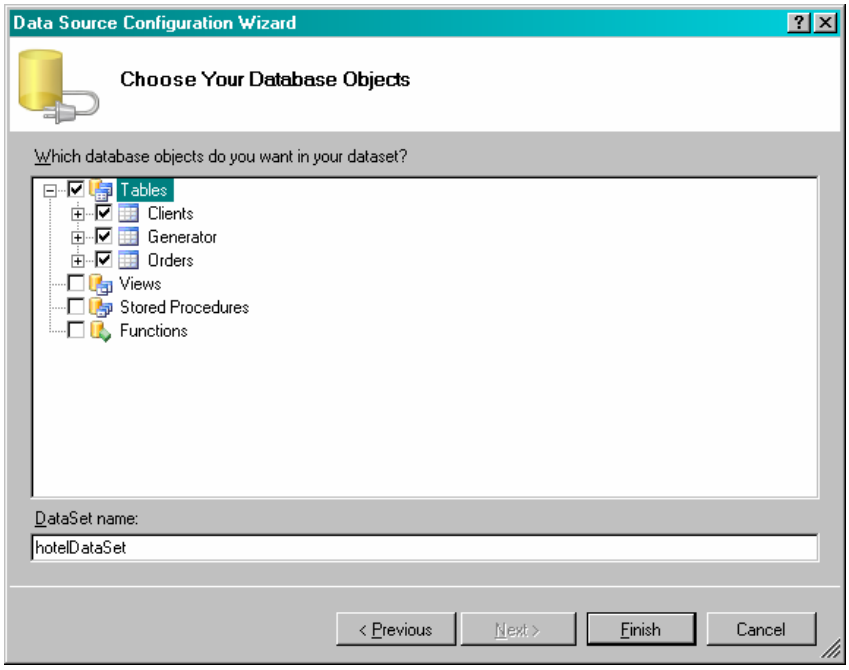


Рисунок 7.10 – Выбор структур для источника данных

Просмотреть созданный источник данных можно следующим образом: выбрать в главном меню «Data\Show Data Sources». После чего отобразится созданная структура hotelDataSet, в контекстном меню которой можно выбрать пункт «Edit DataSet with Designer» для редактирования (рисунок 7.11).

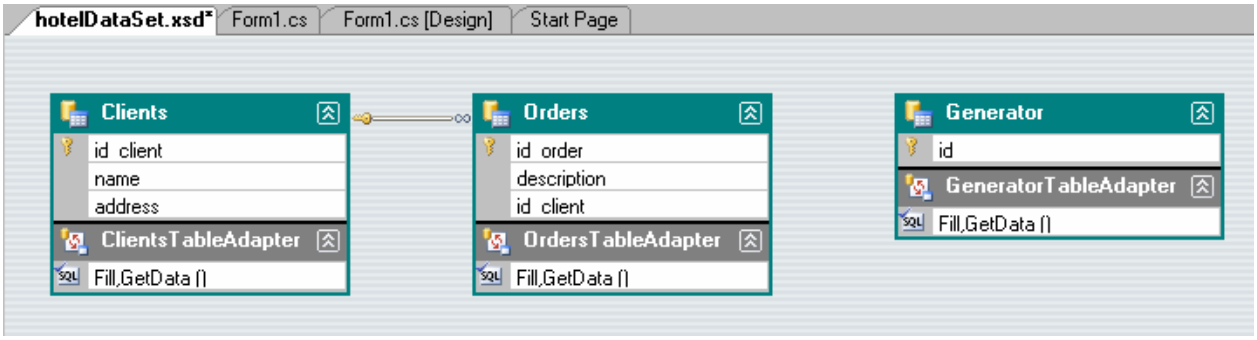


Рисунок 7.11 – Источник данных hotelDataSet

7.3 Проектирование интерфейса приложения

Добавим двух клиентов в таблицу clients. Для этого в главном меню «Server Explorer\Tables» выберем таблицу clients и в контекстном меню укажем пункт «Show Table Data». После этого нужно заполнить соответствующие столбцы (Рисунок 7.12)

	id_client	name	address
	1	Иванов	Барнаул
	2	Петров	Бийск
▶*	NULL	NULL	NULL

Рисунок 7.12 – Добавление строк в таблицу

Теперь выберем в DataSources таблицу clients и перенесем столбцы на созданную форму. После этого запустим проект.

В данной форме можно осуществлять просмотр, редактирование, добавление и удаление столбцов. Код для соответствующих действий генерируется автоматически, поэтому он рассчитан на простые сценарии для несложных приложений. Очистим созданную форму, сгенерированный код и удалим данные из таблицы clients для программирования базы данных.

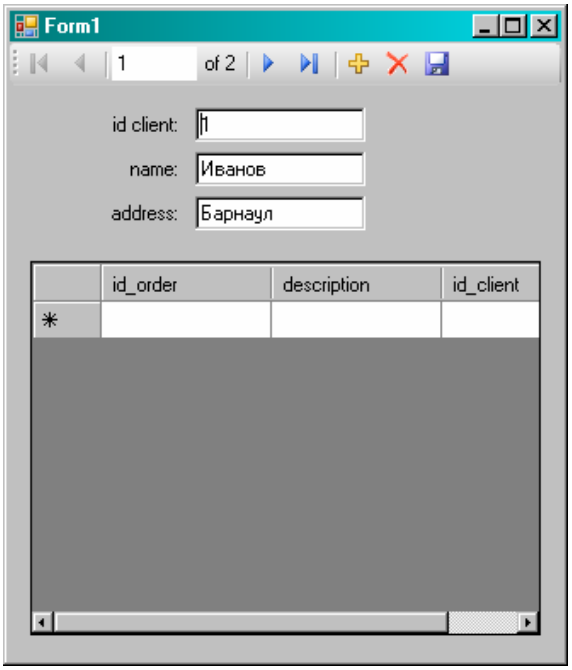


Рисунок 7.13 – Создание формы «Клиенты»

7.4 Программирование базы данных с помощью C#

Объектная модель ADO.NET обеспечивает структуру для доступа к данным из различных источников. Две главные части ADO.NET – DataSet и .NET Data provider. DataSet – это набор объектов из базы данных. Обычно он состоит из одного или нескольких объектов DataTables, а также может включать в себя объекты DataView и другие. Data provider обеспечивает связь между источником данных и DataSet. Data provider включает в себя объекты: Connection – подключение к источнику, Command – выполнение команд на источнике, DataReader – обеспечивает поток данных с источника, DataAdapter – обеспечивает связь между DataSet и Data provider, а также ответственен за применение изменений, которые вносятся в источник данных. Объект DataSet позволяет хранить информацию в рассоединенной среде. После установления соединения с базой данных в объект DataSet обычно помещаются данные, и пользователь работает уже с ними. Если пользователь внес изменения в DataSet, то для записи этих изменений в DataSet соединение может быть установлено повторно. С помощью DataSet клиентское звено приложения получает возможность читать и обрабатывать его содержимое, будучи отсоединенным от источника данных, чтобы затем с помощью адаптера направить измененные данные обратно.

Важным в программировании баз данных с помощью ADO.NET является понятие поставщик данных – это набор типов, обеспечивающих базовые функ-

циональные возможности для связи с СУБД. Рассмотрим базовые классы поставщика данных: DbConnection – обеспечивает соединение с хранилищем данных и отключение от него; DbCommand – обеспечивает SQL-запрос; DbDataReader – обеспечивает доступ к данным в режиме только чтение; DbDataAdapter – обеспечивает обмен объектами DataSet между вызывающей стороной и местом хранения данных, содержит набор из четырех команд, используемых для выборки, вставки, удаления и обновления информации из хранилища данных; DbParameter – именованный параметр параметризованного запроса; DbTransaction – выполняет транзакцию базы данных.

Рассмотрим более подробно каждый из классов поставщика данных. DbConnection реализуется объектом соединения поставщика данных. Данный класс содержит множество членов, используемых для настройки соединения с конкретным хранилищем данных и получения объекта транзакции поставщика данных.

Использование членов DbTransaction позволяет осуществлять взаимодействие с сеансом транзакции и хранилищем данных.

Рассмотрим использование DbConnection и DbTransaction на примере поставщика данных Microsoft SQL Server:

Листинг 7.1 – Использование поставщика данных Microsoft SQL Server

```
// создание объекта соединения поставщика данных
SqlConnection connection = new SqlConnection(ConnectionString);
// открытие соединения
connection.Open();
// начало транзакции
SqlTransaction transaction = connection.BeginTransaction();
…
// принять изменения
transaction.Commit();

// не принимать изменения
// transaction.RollBack();

// закрытие соединения
connection.Close();
```

Объекты класса DbCommand позволяют программно обрабатывать SQL-операторы, хранимые процедуры и параметризованные запросы. DbDataParameter позволяет представлять параметры SQL-команды в виде специальных объектов параметров ADO.NET.

Адаптеры данных используются для извлечения объектов DataSet из хранилища данных и отправки их в хранилище. DbDataAdapter определяет набор свойств, используемых для поддержки SQL-операторов в операциях выборки, вставки, обновления и удаления данных.

Листинг 7.2 – Код операций выборки, вставки, обновления и удаления

```
// создание адаптера
SqlDataAdapter dataAdapter = new SqlDataAdapter();
// определение оператора выбора для адаптера
dataAdapter.SelectCommand =
new SqlCommand("SELECT* FROM Orders", connection, transaction);
// определение оператора вставки для адаптера
dataAdapter.InsertCommand =
new SqlCommand("INSERT INTO Orders (id_order, description,id_client)
VALUES (@p1,@p2,@p3)", connection, transaction);
// создание именованного параметра целого типа для команды вставки
SqlParameter Parametr = new SqlParameter("@p1", SqlDbType.Int);
// выбор столбца-источника из DataSet, соответствующего параметру
Parametr.SourceColumn = "id_order";
// добавление созданного параметра в команду на вставку
dataAdapter.InsertCommand.Parameters.Add(Parametr);
…
// определение оператора обновления для адаптера
dataAdapter.UpdateCommand =
new SqlCommand("UPDATE Orders SET description= @p2, id_client=@p3
WHERE (Orders.id_order=@p1)", connection, transaction);
// создание именованных параметров для команды обновления
…
// определение оператора удаления для адаптера
dataAdapter.DeleteCommand =
new SqlCommand("DELETE FROM Orders
WHERE (Orders.id_order = @p1)", connection, transaction);
// создание именованных параметров для команды удаления
…
// вызов оператора Select для таблицы Orders
dateAdapter.Fill(dataSet, "Orders");
// вызов операторов Insert, Update, Delete
dateAdapter.Update(dataSet, "Orders");
```

Компонент DbDataAdapter определяет ключевую функцию адаптера: способность переносить объекты DataSet из приложения вызывающей стороны в хранилище данных и обратно, используя методы Fill( ) и Update( ).

Для корректной работы с данными через ADO.NET приложение должно обеспечивать выполнение следующих действий:

1) клиент запрашивает данные;

2) создаются объекты Connection и DataAdapter, происходит установка соединения;

3) объект DataSet заполняется данными через DataAdapter и соединение с источником данных разрывается;

4) объект DataSet предоставляется клиенту. Клиент просматривает полученные данные и вносит в них изменения;

5) для сохранения изменений на источнике данных объект «DataAdapter» опять обращается к объекту Connection, затем открывается соединение с источником и в источник вносятся изменения.

Чтобы реализовать данные действия, создадим класс для каждой таблицы из исходной базы данных. Для этого в контекстном меню проекта нужно выбрать «Add\New Item», далее выбрать класс и ввести имя OrdersAccessor.cs.

Листинг 7.3 – Код класса OrdersAccessor.cs

```
// конструктор
public OrdersAccessor()
{
}
// заполнение таблицы Orders набора данных DataSet
   результатами выполнения метода SelectCommand объекта DataAdapter
public void Fill(DataSet dataSet, SqlConnection connection,
SqlTransaction transaction)
{
SqlDataAdapter dateAdapter = CreateAdapter(connection, transaction);
dateAdapter.Fill(dataSet, "Orders");
}
// вызов операторов Insert, Update, Delete для каждой вставленной,
   обновленной и измененной строки таблицы Orders
public void Update(DataSet dataSet, SqlConnection connection,
SqlTransaction transaction)
{
SqlDataAdapter dateAdapter = CreateAdapter(connection, transaction);
dateAdapter.Update(dataSet, "Orders");
}
```

```
// создание объекта DataAdapter с определением
   команд Insert, Update и Delete
private SqlDataAdapter CreateAdapter(SqlConnection connection,
SqlTransaction transaction)
{
…
}
```

Для реализации значений первичных ключей при добавлении строк в какую-либо таблицу необходимо создать класс GeneratorAccessor. Для этого была разработана таблица Generator, поле id которой содержит все существующие значения первичных ключей. При добавлении строки в какую-либо таблицу необходимо выбрать максимальное значение из id и добавить это значение, увеличенное на 1. Если таблица Generator не содержит данных, то необходимо вставить 0. Чтобы выполнить команду выбора максимального значения необходимо выполнить метод команды ExecuteScalar( ), который возвращает единственное значение. Чтобы выполнить команду вставки нового значения необходимо выполнить метод команды ExecuteNonQuery( ), который направляет текст команды в хранилище.

Для создания и управления объектами описанных классов используется класс BusinessLogic.cs. В его конструкторе создаются объекты существующих классов. Для получения первичного ключа используется метод GetNextId, который возвращает GeneratorAccessor.GetNextId( ).

Для заполнения DataSet создается метод, который в свою очередь создает DataSet и заполняет его данными с помощью вызова clients.Fill(dataSet, connection, transaction) и orders.Fill(dataSet, connection, transaction);

Для принятия изменений, произошедших в DataSet, используются методы UpdateClients и UpdateOrders, каждый из которых содержит метод для принятия изменений соответствующей таблицы: orders.Update(dataSet, connection, transaction) и clients.Update(dataSet, connection, transaction). Таким образом, класс BusinessLogic реализует получение данных из базы данных и передачу изменений в базу данных.

Перейдем к главной форме приложения. В конструкторе данной формы необходимо создать объект класса hotelDataSet и объект класса BusinessLogic.

Создадим на форме панель tabControl1 с двумя вкладками – заказы и клиенты. В каждую вкладку добавим dataGridView1 и dataGridView2 для отображения данных из таблиц orders и clients.

Свойство VirtualMode для dataGridView1 необходимо установить в значение TRUE. DataGridView поддерживает специальный виртуальный режим отображения данных. Основная идея такого режима заключена в том, что внутри панели закладок не хранится никаких данных. Вместо этого DataGridView генерирует события, в обработчиках которых можно предоставлять данные, или наоборот, полу-

чить данные, введенные пользователем. Подавать данные можно только свободным колонкам.

Когда в DataGridView требуется узнать данные ячейки (например, для прорисовки или подбора оптимальной ширины колонки), в виртуальном режиме генерируется событие CellValueNeeded. Параметр «e» (тип DataGridViewCellValueEventArgs) имеет три свойства: RowIndex, ColumnIndex и Value. Первые два из них доступны только для чтения и предоставляют целочисленный индекс строки и столбца. Свойство Value имеет тип Object и доступно для записи. Собственно, цель обработчика события заключается в том, чтобы на основе индексов столбца и строки вычислить значение ячейки и подставить его в свойство Value. Именно это значение и будет использовать DataGridView в качестве значения ячейки.

При загрузке главной формы необходимо передать данные в DataSet с помощью метода ds.Merge(bl.Get( )).

Для связи dataGridView с соответствующей таблицей необходимо свойству DataSource указать DataSet.<имя таблицы>. Если DataGridView связан через свойство DataSource с источником данных, то по умолчанию выполняется следующее: каждый столбец, получаемый из источника данных, вызовет добавление соответствующего столбца в tabControl1, названия столбцов источника отобразятся в заголовках столбцов, если пользователь кликнет по заголовку столбца, строки будут автоматически отсортированы.

Далее необходимо убрать ненужные для отображения столбцы (первичный ключ, внешний ключ), добавить заголовки для остальных столбцов и добавить столбцы вместо внешнего ключа. Также для настройки внешнего вида DataGridView необходимо изменить следующие свойства, представленные в листинге 7.4.

Листинг 7.4 – Настройка интерфейса DataGridView

```
// не отображать значение первичного ключа
dataGridView1.Columns["id_order"].Visible = false;
// для выделения пользователем целой строки
SelectionMode = DataGridViewSelectionMode.FullRowSelect;
//для определения длины столбца по длине ячейки
dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode. DisplayedCells;
// заголовки строк сделать невидимыми
RowHeadersVisible = false;
// невозможность выделения нескольких строк
MultiSelect = false;
// невозможность изменять данные вручную
```

```
ReadOnly = true;
// невозможность добавлять строки вручную
AllowUserToAddRows = false;
// невозможность удалять строки вручную
AllowUserToDeleteRows = false;
// невозможность изменять размеры строк
AllowUserToResizeRows = false;
// невозможность изменять размеры столбцов
AllowUserToResizeColumns = false;
```

Таким образом, при загрузке приложения должны отобразиться все данные в dataGridView1 и dataGridView2. Для проверки внесем информацию в базу данных (рисунки 7.14 и 7.15).

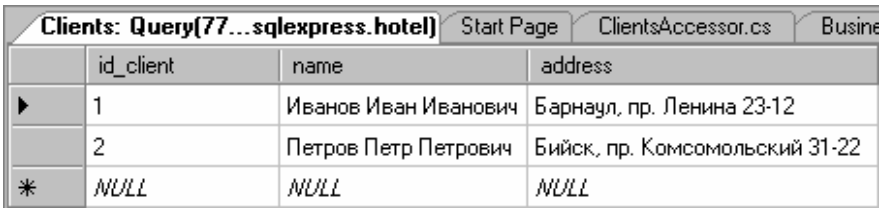


Рисунок 7.14 – Добавление клиентов в базу данных

Рисунок 7.15 – Добавление заказов в базу данных

Параллельно необходимо добавить строку id = 3 в таблицу generator. Интерфейс созданных окон приложения для работы с таблицами orders и clients представлен на рисунке 7.16 и 7.17 соответственно.

Далее реализуем добавление, изменение и удаление данных на примере таблицы orders. Поместим на форму кнопку «Добавить» и напишем обработчик события – нажатие на эту кнопку. При добавлении новой строки нужно создать новый код: hotelDataSet.OrdersRow or = ds.Orders.NewOrdersRow(). После этого необходимо инициализировать ее пустыми значениями. Далее создадим форму для добавления и редактирования и перегрузим ее метод ShowDialog со ссылочными параметрами типа hotelDataSet, hotelDataSet.OrdersRow и обычным параметром типа bool, который возвращает значение типа DialogResult. Форма добавления и редактирования приведена на рисунке 7.18.

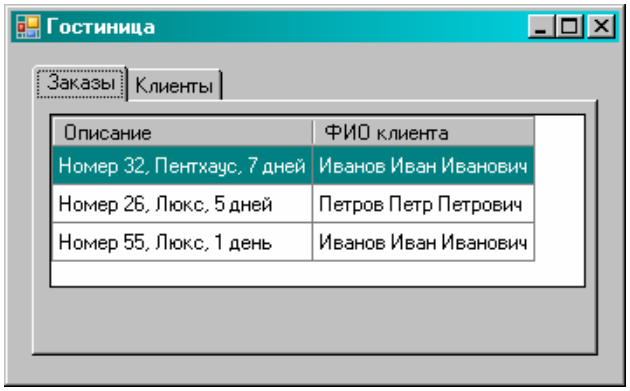


Рисунок 7.16 – Главное окно программы с открытой вкладкой заказы

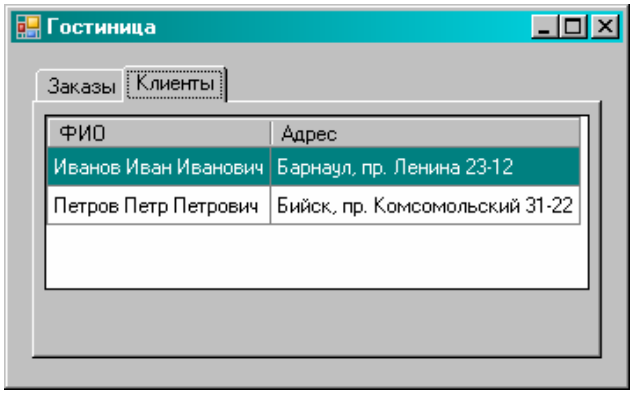


Рис. 7.17 – Главное окно программы с открытой вкладкой клиенты

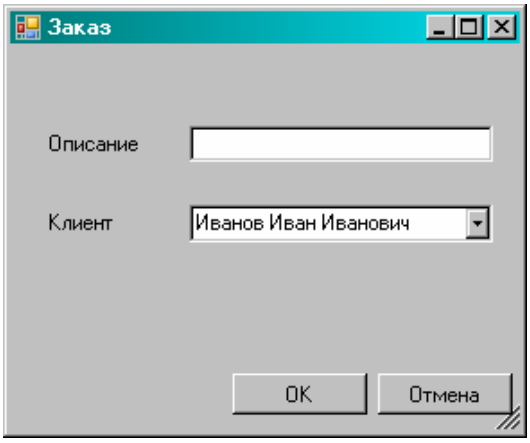


Рисунок 7.18 – Интерфейс формы добавления и редактирования строк

Для того, чтобы связать свойство id_client объекта OrdersRow с элементом управления ComboBox, нужно использовать программный код, представленный в листинге 7.5.

Листинг 7.5 – Настройка элемента ComboBox

```
comboBox1.DataSource = ds.Clients;
comboBox1.DisplayMember = "name";
comboBox1.ValueMember = "id_client";
comboBox1.DataBindings.Add("SelectedValue", or, "id_client");
```

Параметр типа bool используется для того, чтобы определить, вставляются данные или редактируются. Если редактируются, то необходимо выполнить or.BeginEdit( ).

Далее нужно вызвать метод ShowDialog( ) для данной формы. Если пользователь нажал на кнопку «OK», то необходимо вернуть DialogResult.OK, иначе необходимо вернуть DialogResult.Cancel. Перед закрытием окна редактирования необходимо выполнить команду or.EndEdit( ) или or.CancelEdit( ).

Если пользователь вернул DialogResult.OK, то нужно сгенерировать новый id_order, добавить данные в таблицу orders, обновить данные с помощью метода UpdateOrders класса BusinessLogic, принять изменения в hotelDataSet с помощью метода ds.AcceptChanges( ) и освободить ресурсы созданной формы.

Для редактирования строки таблицы необходимо получить текущую строку hotelDataSet.OrdersRow, которую пользователь пытается изменить. Из-за специфики ADO.NET, поскольку работа идет с локальной копией данных, возникает проблема получения текущей (выделенной в объекте отображения) строки данных – DataRow. Проблема в том, что объект отображения может сортировать или фильтровать отображаемые строки, нарушая их порядок.

CurrencyManager управляет всеми связями (объектами Bindings) для одного источника данных. В классе BindingContext определен индексатор, который дает доступ к элементам коллекции – объектам CurrencyManager (листинг 7.6).

Листинг 7.6 – Настройка связей базы данных с помощью CurrencyManager

```
// создание объекта класса CurrencyManager
   для управления связями от ds.Orders
CurrencyManager cm = (CurrencyManager)this.BindingContext
[dataGridView1. DataSource, dataGridView1.DataMember];
// получение текущей строки DataRowView
DataRowView drv = (DataRowView)cm.Current;
// получение текущей строки hotelDataSet.OrdersRow
hotelDataSet.OrdersRow or = (hotelDataSet.OrdersRow)(drv).Row;
```

Полученную строку необходимо передать в форму для редактирования. Для удаления текущей записи необходимо получить текущую строку hotelDataSet.OrdersRow и, вызвав метод Delete, удалить ее.

На этом разработка приложения, работающего с базой данных и реализующего основные функции работы с данными, завершена.

Тезаурус

Поставщик данных, класс, метод, свойство, адаптер, хранилище данных.

Контрольные вопросы

1) Что такое поставщик данных .NET?
2) Какой класс используется для соединения с базой данных?
3) Для чего нужно сохранять значение ConnectionString?
4) Каким методом класса DataAdapter можно осуществить выборку и перенос результата в объект DataSet для последующей автономной работы?
5) Какой класс выступает в качестве контейнера объектов DataTable?
6) Приведите пример отсоединенных и подсоединенных классов. В чем их отличие?
7) Какие свойства DatatAdapter служат для обмена данными между хранилищем и объектом DataSet?
8) Для чего создается таблица Generator?
9) Какое событие DataGridView срабатывает для пустых ячеек при установленном виртуальном режиме?
10) Как можно связать элемент TextBox с источником данных?

Список рекомендуемой литературы

1) Троэлсен, Э. Язык программирования C# 2005 и платформа .NET 2.0 [Текст] / Э. Троэлсен. – М. : Вильямс, 2007. – 1155 с.
2) Сеппа, Д. Программирование на ADO.NET 2.0 [Текст] / Д. Сеппа. – СПб. : Питер, 2007. – 785 с.
3) Купцевич, Ю. В. Альманах программиста: ADO.NET, SQL Server, доступ к данным из приложения [Текст] / Ю. В. Купцевич. – М. : Русская редакция, 2003. – 403 с.
4) Нэш, Т. C# 2008 ускоренный курс для профессионалов [Текст] / Т. Нэш. – М. : Вильямс, 2008. – 576 с.
5) Лабор, В. В. C# создание приложений для Windows [Текст] / В. В. Лабор. – Минск : Харвест, 2003. – 385 с.
6) Дейтел, Х. C# в подлиннике. Наиболее полное руководство [Текст] / Х. Дейтел. – СПб. : BHV, 2006. – 1056 с.
7) Понамарев, В. А. Программирование на C++/C# в Visual Studio .NET [Текст] / В. А. Понамарев. – СПб. : BHV, 2004. – 352 с.

ГЛАВА 8
РАЗРАБОТКА БАЗ ДАННЫХ ДЛЯ ГЛОБАЛЬНОЙ СЕТИ ИНТЕРНЕТ

В настоящей главе рассматриваются вопросы создания баз данных MySQL с помощью системы управления базами данных PHPMyAdmin и разработки клиентских приложений на языке программирования PHP.

8.1 Разработка структуры базы данных MySQL

С помощью таких средств программирования как C# можно создавать базы данных и приложения для глобальной сети Интернет, однако, процесс их создания трудоемкий и требует высокой квалификации. Для разработки профессиональных баз данных для глобальной сети Интернет с наименьшими затратами и наибольшей эффективностью можно использовать системы управления базами данных на основе MySQL, которые очень популярны среди разработчиков из-за простоты работы и бесплатной лицензией использования.

Разработка базы данных начинается с ее регистрации на сервере, причем администратор выдает разработчику название базы данных и пароль доступа к ней. Для проектирования структуры базы данных необходимо загрузить среду PHPMyAdmin 2.6.1 и выбрать базу данных MySQL 5.0.45. Система управления базами данных PHPMyAdmin на сегодняшний день является одним из самых популярных инструментов работы с базами данных MySQL в глобальной сети Интернет и присутствует практически на всех серверах.

Чтобы создать таблицу MySQL необходимо ввести название таблицы и количество содержащихся в ней столбцов (рисунок 8.1).

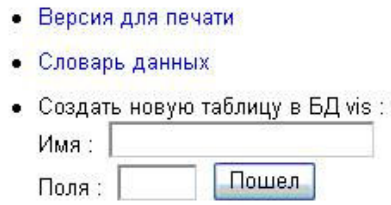


Рисунок 8.1 – Разработка таблицы

Далее необходимо определить название столбцов таблицы и тип содержащихся в них данных, ввести значения по умолчанию, дополнительные атрибуты, если столбец может содержать нулевые значения, то указать это в соответствующем поле (рисунок 8.2). Здесь же нужно установить ключевые столбцы таблицы: первичный, уникальный, индексный ключ.

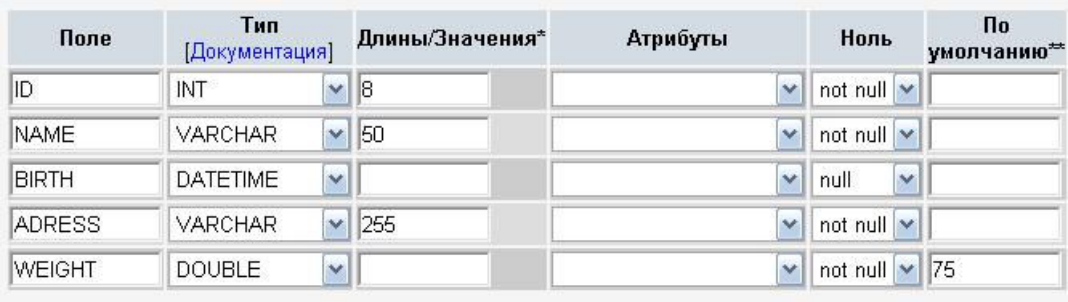


Рисунок 8.2 – Создание столбцов таблицы

После этого нужно ввести комментарий к таблице, определить ее тип и нажать кнопку «Сохранить» (рисунок 8.3). Рекомендуется использовать тип таблицы MyISAM, который является наиболее производительным и быстрым при работе с данными.

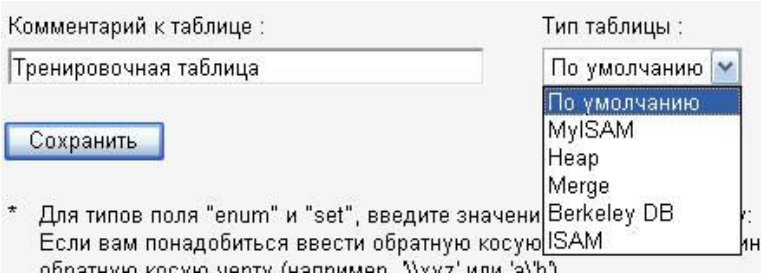


Рисунок 8.3 – Сохранение таблицы

В процессе работы структуру таблицы можно в любой момент изменить.

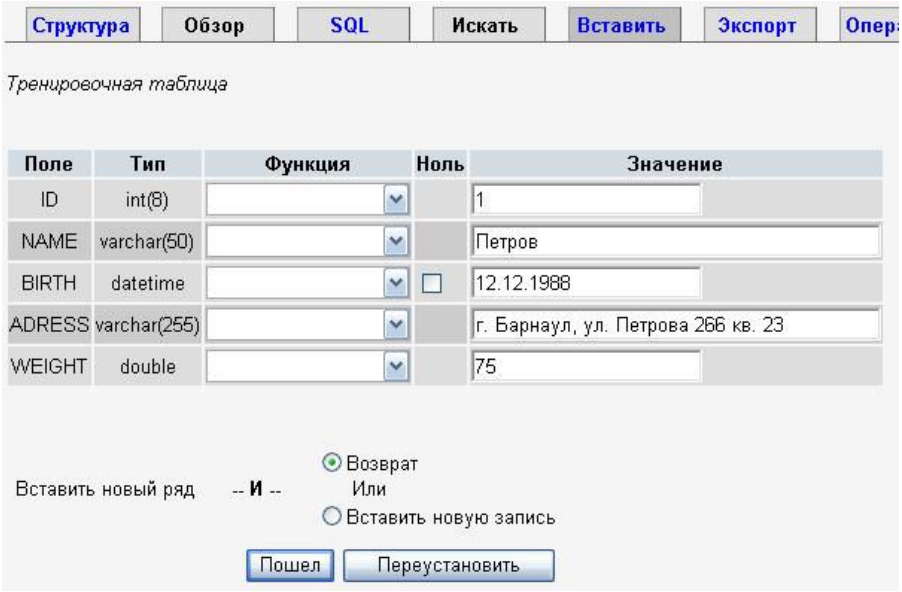


Рисунок 8.4 – Вставка данных в таблицу

Чтобы ввести данные в таблицу необходимо выбрать соответствующую таблицу и перейти на вкладку «Вставить». Далее заполнить нужные поля значениями, если это необходимо, определить агрегирующие функции и нажать на кнопку «Пошел» (рисунок 8.4).

После вставки данных в таблицу их можно просмотреть. Для этого необходимо перейти на вкладку «Обзор», выбрать количество строк, отображаемых на экране (рисунок 8.5). Затем можно выполнить необходимые действия над данными или группой данных: добавление, редактирование или удаление.

Рисунок 8.5 – Просмотр данных

Для того, чтобы выполнить операции над таблицей в целом, необходимо перейти на вкладу «Удалить» или «Очистить», в первом случае таблица удалиться, во втором – уничтожатся все существующие в ней данные.

Чтобы экспортировать таблицу необходимо перейти на вкладку «Экспорт», указать опции экспорта данных: тип экспорта – SQL; структура – добавить значение AUTO_INCREMENT и установить обратные кавычки в названиях таблиц и полей; данные – полная вставка; указать шаблон имени файла и нажать на кнопку «Пошел».

После создания структуры базы данных и ее информационного наполнения необходимо разработать запросы и создать клиентские приложения.

8.2 Разработка и тестирование запросов MySQL

Для того, чтобы разработать и протестировать запрос в среде PHPMyAdmin, необходимо перейти на вкладку «SQL», ввести разработанный запрос, нажать на кнопку «Пошел» и получить результат запроса.

Рисунок 8.6 – Разработка и тестирование запросов

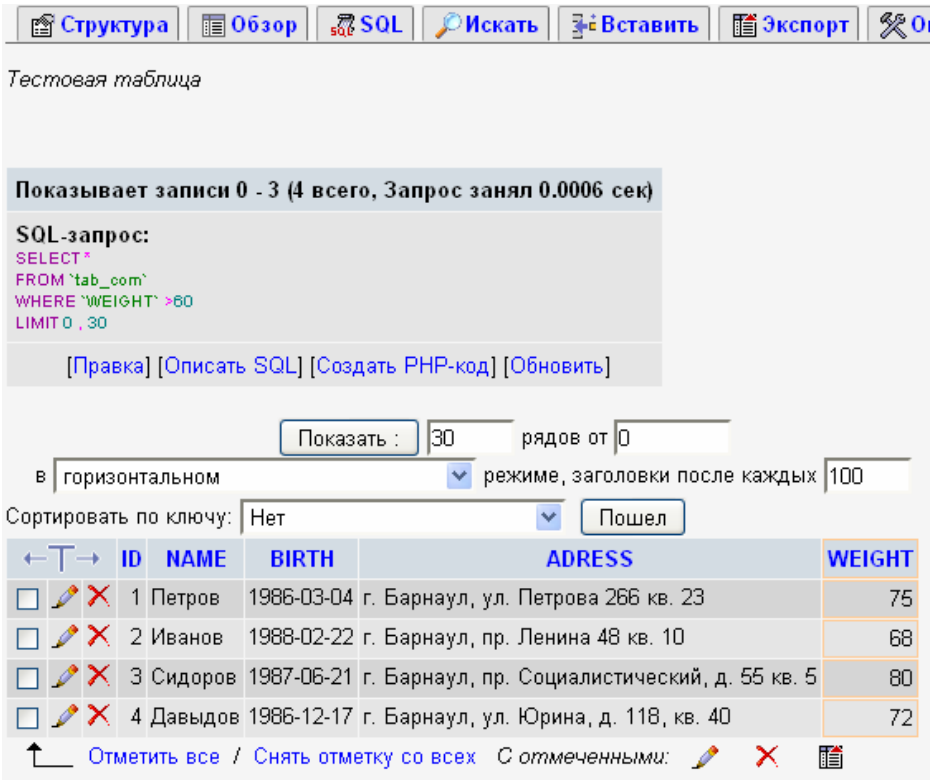


Рисунок 8.7 – Вывод результата выполнения запроса

Запросы можно загружать из текстовых файлов, которые затем необходимо запускать на выполнение. После выполнения запроса появляется результирующая выборка. Далее можно отредактировать запрос, на его основе сформировать PHP-код для внедрения в приложение (рисунок 8.7).

8.3 Проектирование клиентского приложения с помощью PHP

С помощью языка SQL пользователь формирует запрос, а система управления базами данных выполняет ее и возвращает результат пользователю. Использование в качестве интерфейса между пользователем и системой управления базами данных языка SQL накладывает некоторые требования на квалификацию пользователя. Он должен знать язык SQL и уметь им пользоваться. Большинство пользователей не являются специалистами в области информационных технологий, поэтому для облегчения их доступа к информации разрабатываются приложения, имеющие более простой интерфейс. Для построения такого интерфейса используются, как правило, визуальные элементы, которые подсказывают пользователю, какой результат он получит, если будет использовать этот элемент интерфейса.

Большинство приложений для глобальной сети Интернет создаются с помощью серверного языка сценариев PHP, предназначенного для встраивания кода в HTML-файлы с помощью специальных тэгов. В PHP имеются собственные средства поддержки баз данных. Благодаря хорошим характеристикам и обширному набору стандартных интерфейсных функций MySQL стала самым популярным средством для работы с базами данных в PHP.

Общая последовательность действий PHP при взаимодействии с сервером MySQL выглядит следующим образом:

1) установить соединение с сервером MySQL. Если попытка завершается неудачей, вывести соответствующее сообщение и завершить процесс;

2) выбрать базу данных сервера MySQL. Если попытка выбора завершается неудачей, вывести соответствующее сообщение и завершить процесс. Допускается одновременное открытие нескольких баз данных для обработки запросов;

3) обработать запросы к выбранной базе данных;

4) после завершения обработки запросов закрыть соединение с сервером баз данных.

Подключиться к серверу MySQL из PHP несложно. Достаточно вызвать функцию mysql_pconnect() с соответствующими аргументами. Функция возвращает идентификатор соединения, требуемый для большинства остальных функций.

После подключения к серверу баз данных можно посылать ему запросы с помощью функции mysql_query( ). Эта функция принимает в качестве аргумента текст SQL-инструкции и идентификатор соединения, полученный от функции mysql_pconnect( ), после чего возвращает идентификатор результирующего набора данных. В случае ошибки возвращается нуль.

Важным моментом является использование подстановочных столбцов, в MySQL подстановочные столбцы определяются в запросе, где вместо первичного ключа в главной таблице подставляется соответствующее значение из подчиненной таблицы.

Функция mysql_result( ) извлекает значение одной ячейки, но гораздо удобнее пользоваться функцией mysql_fetch_row( ) или mysql_fetch_array( ). Все они возвращают следующую строку из указанного набора, что позволяет эффективно применять их в цикле while.

Запросы на вставку или обновление данных не возвращают наборы данных. Они тоже выполняются с помощью функции mysql_query( ). Если требуется узнать число добавленных или измененных строк, нужно воспользоваться функцией mysql_affected_rows( ).

Рассмотрим в качестве примера базу данных «Домашняя библиотека», которая содержит две таблицы: жанр (about) и книги (book), связанных между собой посредством первичного ключа about.id и внешнего ключа book.type (рисунок 8.8).

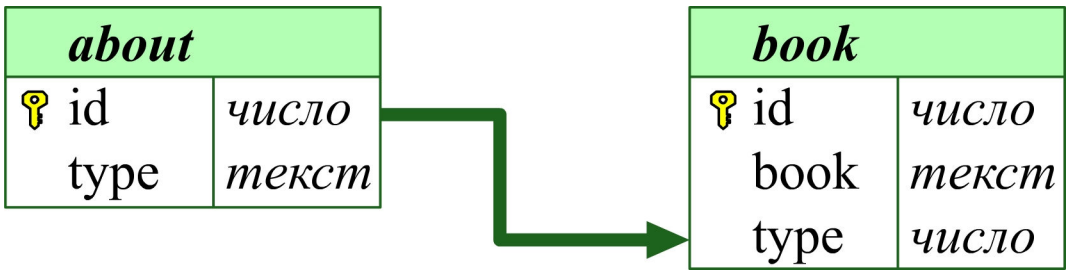


Рисунок 8.8 – Структура базы данных «Библиотека»

Для того, чтобы вывести информацию об имеющихся в базе данных книгах, разработаем клиентское приложение. Для этого необходимо создать главную страницу book.php, в которой будет представлена вся информация из таблицы book, причем, рядом с каждой строкой расположим две ссылки «изменить» и «удалить» для модификации строк, при нажатии на эти ссылки в соответствующие формы передается номер текущей строки базы данных. Дополнительно создадим ссылку «Добавить» для вставки новых строк. В начале каждой формы осуществляется соединение с сервером и базой данных, а также проверяется корректность этих соединений (листинг 8.1).

Листинг 8.1 – Код страницы index.php

```
<html>
<head>
<title>Книги</title>
</head>
```

```
<body>
<?
if(!($dbLink = mysql_pconnect("localhost","reader","zrx34cvCC")))
{
print("Не могу соединиться с сервером!<br\n>");
print(mysql_errno().":".mysql_error()."<br>\n");
exit;
}
if(!mysql_select_db("test",$dbLink))
{
print("Не могу соединиться с базой данных!<br\n>");
print(mysql_errno().":".mysql_error()."<br>\n");
exit;
}
$Query = "SELECT book.id AS i, book.name AS b, about.type AS t
FROM book, about WHERE book.type = about.id";
if(!($dbResult = mysql_query($Query,$dbLink)))
{
print("Не могу выполнить запрос!<br\n>");
print(mysql_errno().":".mysql_error()."<br>\n");
exit;
}
?>
<div align="center"><strong>База данных Домашняя
библиотека</strong></div>
<div align="center"><a href=book_insert.php>добавить книгу</a>
<table border='1' cellspacing='0'>
<tr><td><strong>Книга</strong></td>
<td><strong>Жанр</strong></td>
<td>&copy;</td><td>&#8212;</td></tr>
<?
while(list($i, $b, $t) = mysql_fetch_row($dbResult))
{
print("<tr>\n");
printf("<td><div align='left'>%s</div></td><td>
<div align='left'>%s</div></td>",$b,$t);
printf("<td><a href=\"book_update.php?id=%s\">изменить</a></td>",$i);
printf("<td><a href=\"book_delete.php?id=%s\">удалить</a></td>",$i);
print("</tr>\n");
}
?>
```

```
</table>
</div>
</body>
</html>
```

После этого нужно создать страницу book_delete.php. Данная страница по полученному идентификатору записи посредством запроса удаляет соответствующую запись (листинг 8.2).

Листинг 8.2 – Код страницы book_delete.php

```
<html>
<head>
<title>Изменение книги</title>
</head>
<body>
<div align="center"><strong>База данных Домашняя библиотека</strong>
<?
$dbLink = mysql_pconnect("localhost","reader","zrx34cvCC");
mysql_select_db("test",$dbLink);
$Query = "DELETE FROM book WHERE id=".$id;
if(!($dbResult = mysql_query($Query,$dbLink)))
{
print("Не могу удалить записи!<br\n>");
print(mysql_errno().":".mysql_error()."<br>\n");
}
print("<p>Запись удалена!<p>");
?>
<a href=index.php>назад</a>
</div>
</body>
</html>
```

Далее необходимо спроектировать страницу book_insert.php. На этой странице разместим в форме поле для ввода названия книги (идентификатор заполняется автоматически), ниспадающий список со всеми имеющимися в базе данных жанрами, а также кнопку для подтверждения изменения сстроки. Значения в поле для ввода и активное значение в ниспадающем списке остаются пустыми и заполняются пользователями.

Листинг 8.3 – Код страницы book_insert.php

```
<html>
<head>
<title>Добавление книги</title>
</head>
<body>
<?
$dbLink = mysql_pconnect("localhost","reader","zrx34cvCC");
mysql_select_db("test",$dbLink);
$Query = "SELECT id, type FROM about ORDER BY id";
$dbResult = mysql_query($Query,$dbLink);
$str="";
while(list($i, $about) = mysql_fetch_row($dbResult))
{
$str.="<option value=".$i.">".$about."</option>";
}
?>
<div align="center"><strong>База данных Домашняя
библиотека</strong></div>
<form name="form1" method="post" action="book_insert_post.php">
<table width="100%" border="0" align="center" cellspacing="5">
<tr>
<td><strong>Книга</strong></td>
<td><input name="name" type="text" id="name" value=""></td>
</tr>
<tr>
<td><strong>Жанр</strong></td>
<td><select name="type" size="1" id="type">
<? echo $str;?>
</select></td>
</tr>
</table>
<div align="center">
<input type="submit" name="Submit" value="Добавить">
</div>
</form>
<div align="center"><a href=index.php>назад</a>
</div>
</body>
</html>
```

При нажатии на кнопку «Добавить» страницы book_insert.php запускается страница book_insert_post.php, все данные с формы передаются методом POST. После этого выполняется запрос на добавление записи:

Листинг 8.4 – Код страницы book_insert_post.php

```
<html>
<head>
<title>Добавление книги</title>
</head>
<body>
<div align="center"><strong>База данных Домашняя библиотека</strong>
<?
$dbLink = mysql_pconnect("localhost","reader","zrx34cvCC");
mysql_select_db("test",$dbLink);
$Query = "INSERT INTO book (name,type) VALUES ('".$name."',".$type.")";
if(!($dbResult = mysql_query($Query,$dbLink)))
{
print("Не могу добавить запись!<br\n>");
print(mysql_errno().":".mysql_error()."<br>\n");
}
print("<p><div align='center'>Запись добавлена!</div><p>");
?>
</div>
<div align="center"><a href=index.php>назад</a>
</div>
</body>
</html>
```

После этого необходимо создать страницу book_update.php. На этой странице разместим в форме поля для ввода идентификатора (его изменять нельзя, так как он является первичным ключом) и названия книги, ниспадающий список со всеми имеющимися в базе данных жанрами, значения в ниспадающем списке заполняются с помощью запроса, а также кнопку для подтверждения изменения записи.

Нужные значения в поля для ввода и активное значение в ниспадающем списке заполняются с помощью запроса, в котором явно указывается номер идентификатора редактируемой строки.

Листинг 8.5 – Код страницы book_insert_post.php

```
<html>
<head>
<title>Изменение книги</title>
</head>
<body>
<div align="center"><strong><?
$dbLink = mysql_pconnect("localhost","reader","zrx34cvCC");
mysql_select_db("test",$dbLink);
$Query = "SELECT book.id AS id, book.name AS b, about.type AS t
FROM book, about WHERE book.type = about.id AND book.id=".$id;
$dbResult = mysql_query($Query,$dbLink);
while(list($i, $b, $t) = mysql_fetch_row($dbResult))
{
$idFromDB = $i;
$nameFromDB = $b;
$typeFromDB = $t;
break;
}
$Query = "SELECT id, type FROM about ORDER BY id";
$dbResult = mysql_query($Query,$dbLink);
$str = "";
while(list($i, $about) = mysql_fetch_row($dbResult))
{
$s="";
if($about==$typeFromDB){$s=" selected";}
$str.="<option value=".$i.$s.">".$about."</option>";
}
?>
</div>
<form name="form1" method="post" action="book_update_post.php">
<table width="21%" border="0" align="center" cellspacing="5">
<tr>
<td width="5%"><strong>Номер</strong></td>
<td width="95%"><input name="id" type="text" id="id" value="
<? echo $idFromDB;?>">
</td>
</tr>
<tr>
<td><strong>Книга</strong></td>
```

```
<td><input name="name" type="text" id="name" value="
<? echo $nameFromDB;?>">
</td>
</tr>
<tr>
<td><strong>Жанр</strong></td>
<td><select name="type" size="1" id="type">
<? echo $str;?>
</select></td>
</tr>
</table>
<div align="center">
<input type="submit" name="Submit" value="Изменить">
</div>
</form>
<p align="center"><a href="index.php">назад</a></p>
</body>
</html>
```

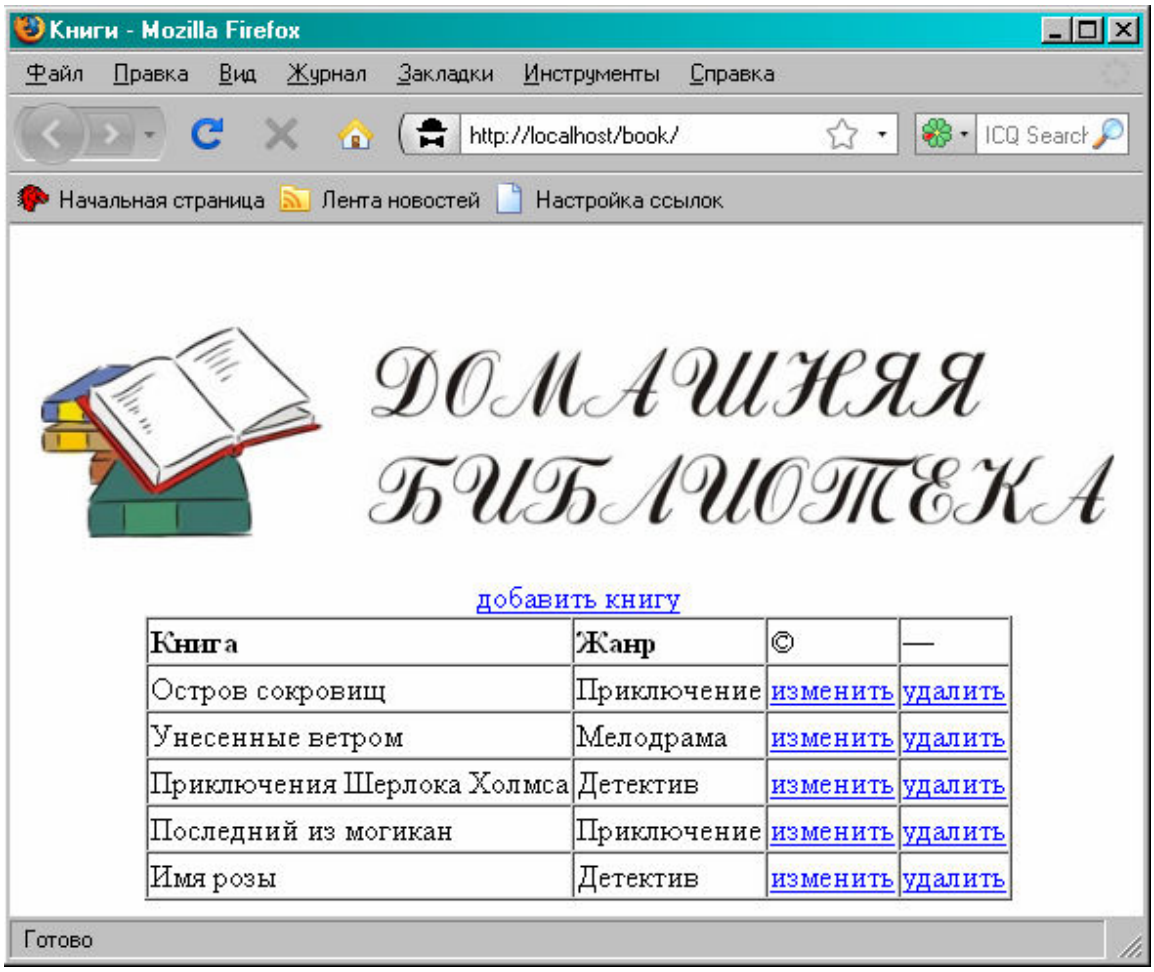
Далее нужно разработать форму book_update_post.php. Здесь методом POST из страницы book_update.php передаются идентификатор измененной строки, название книги и жанр, после чего выполняется запрос на обновление строки.

Листинг 8.6 – Код страницы book_update_post.php

```
<html>
<head>
<title>Изменение книги</title>
</head>
<body>
<div align="center"><strong>База данных Домашняя библиотека</strong>
<?
$dbLink = mysql_pconnect("localhost","reader","zrx34cvCC");
mysql_select_db("test",$dbLink);
$Query = "UPDATE book SET name = '".$name."', type =".$type."
WHERE id=".$id;
if(!($dbResult = mysql_query($Query,$dbLink)))
{
print("Не могу изменить запись!<br\n>");
print(mysql_errno().":".mysql_error()."<br>\n");
}
```

```
print("<p>Запись изменена!<p>");
?>
<a href=index.php>назад</a>
</div>
</body>
</html>
```

Интерфейс главного окна разработанного приложения представлен на рисунке 8.9. Приложение выводит все строки главной таблицы, позволяя каждую из них отредактировать или удалить, также можно добавить новую строку.



Книга	Жанр	©	—
Остров сокровищ	Приключение	изменить	удалить
Унесенные ветром	Мелодрама	изменить	удалить
Приключения Шерлока Холмса	Детектив	изменить	удалить
Последний из могикан	Приключение	изменить	удалить
Имя розы	Детектив	изменить	удалить

Рисунок 8.9 – Интерфейс приложения «Домашняя библиотека»

Тезаурус
Интерфейс, приложение, подстановочный столбец.

Контрольные вопросы
1) Опишите последовательность действия при реализации SQL запроса в приложении, разработанном с помощью языка программирования PHP.
2) Назовите последовательность взаимодействия пользовательского приложения с сервером баз данных.
3) С помощью какой PHP-функции можно вывести результаты выполнения запроса в массив?
4) Как в PHPMyAdmin осуществляется поиск нужной информации?
5) Перечислите типы данных, поддерживаемых MySQL, и области их применения?
6) Для чего необходим экспорт баз данных, и каким образом он осуществляется?
7) В каких случаях при создании столбца указывается дополнительная опция «auto_increment»?
8) Чем отличаются динамические Интернет страницы от статических?
9) Какие существуют методы для передачи данных между страницами?
10) Какой SQL командой задаются привилегии базы данных?

Список рекомендуемой литературы

1) Дюбуа, П. MySQL [Текст] / П. Дюбуа. – М. : Вильямс, 2001. – 816 с.
2) Фролов, А. Г. Базы данных в Интернете [Текст]: практическое руководство по созданию Web-приложений с базами данных / А. Г. Фролов, Г. В. Фролов. – М. : Русская редакция, 2000. – 448 с.
3) Мотев, А. А. Самоучитель. Уроки MySQL [Текст] / А. А. Мотев. – СПб. : BHV, 2006. – 208 с.
4) Аргерих, Л. Профессиональное PHP программирование [Текст] / Л. Аргерих. – М. : Символ-Плюс, 2005. – 1048 с.
5) Чекалов, А. Базы данных от проектирования до разработки приложений [Текст] / А. Чекалов. – СПб. : BHV, 2003. – 384 с.
6) Харрис, Э. PHP/MySQL для начинающих [Текст] / Э. Харрис. – М. : Кудиц-образ, 2005. – 384 с.
7) Веллинг, Л. MySQL [Текст] : учебное пособие / Л. Веллинг. – М. : Вильямс, 2005. – 304 с.

Заключение

Учебное пособие содержит необходимые теоретические сведения для изучения принципов работы современных баз данных. Оно направлено на формирование профессиональных компетенций у студентов инженерной специальности «Информационно-измерительная техника и технологии».

В учебном пособии приведено подробное описание языка структурированных запросов SQL, которое иллюстрируется большим количеством примеров запросов, выполненных на учебной базе данных.

Особое внимание в учебном пособии уделено практической реализации баз данных и приложений с помощью различных технологий и для разных задач. Рассмотрены принципы создания баз данных в Microsoft Office Access 2007, проектирования баз данных и приложений в Microsoft Visual Studio с помощью технологии ADO.NET, а также реализации баз данных для глобальной сети Интернет MySQL с помощью языка программирования PHP.

Авторы благодарят заведующего кафедрой «Информационные технологии» Алтайского государственного технического университета им. И. И. Ползунова, доктора технических наук, профессора Пронина Сергея Петровича за консультации в процессе создания учебного пособия. Авторы также выражают благодарность декану факультета информационных технологий и бизнеса, заведующему кафедрой «Физика и технология композиционных материалов» Алтайского государственного технического университета им. И. И. Ползунова, доктору технических наук, профессору Маркину Виктору Борисовичу за оказание организационной поддержки.

Алфавитный указатель

Агрегируемость 29
Адаптер 106
Администратор базы данных 15
Атрибут 25
Аутентификация 55
База данных 9
Вертикальная выборка 61
Вложение 89
Горизонтальная выборка 63
Группировка 66
Декартово произведение 73
Денормализация 43
Диаграмма 97
Добавление данных 77
Домен 33
Жизненный цикл базы данных 12
Журнал транзакций 47
Запрос 9
Идентификация 55
Избыточность 40
Измерение данных 29
Индекс 19
Инкапсуляция 30
Интервал 85
Интерфейс 12, 88, 128
Класс 107
Ключ внешний 33
Ключ первичный 33
Конструктор 89, 92, 95
Кортеж 32
Макет 93
Многомерность 29
Модель данных 11, 23
Монитор транзакций 50
Наследование 30
Нормализация 40
Нормальная форма 40
Обновление данных 79
Оптимизация баз данных 50
Отношение 28, 32
Отчет 93
Параллелизм 45
Подзапрос 75
Подстановка 89, 121
Подчиненность отношений 34
Полиморфизм данных 30
Поставщик данных 106
Правила Кодда 38
Предикат 63
Представление 9
Реляционная модель 28, 32
Связь 24, 33
Система управления базой данных 9
Слот данных 22
Страница данных 21
Строка данных 22, 32, 59
Структура хранения информации 17
Сущность 24
Транзакция 45
Трафаретный символ 65
Трехуровневая архитектура 15
Удаление данных 80
Уровни изоляции 47
Условный оператор 69
Форма 91
Функция 81
Хранилище данных 102
Хэш-функция 18
Шаблон 88
Шифрование 56

Издано в авторской редакции

Компьютерная верстка, макет: Е. А. Зрюмов
Дизайн обложки: Е. А. Зрюмов, В. В. Немыкин

Издательство Алтайского государственного
технического университета им. И. И. Ползунова
656038, г. Барнаул, пр-т Ленина, 46

Лицензия на издательскую деятельность
ЛР № 020822 от 21.09.98 г.

Подписано в печать 17.03.2010. Формат 60x84 ⅛
Усл. п. л. 15,34. Тираж 100 экз. Заказ 2010 – 120

Отпечатано в типографии АлтГТУ
656038, г. Барнаул, пр-т Ленина, 46
тел. (8-3852) 36-84-61

Лицензия на полиграфическую деятельность
ПЛД №28-35 от 15.07.97 г.