Л. В. Рудикова

БАЗЫ ДАННЫХ
РАЗРАБОТКА ПРИЛОЖЕНИЙ
ДЛЯ СТУДЕНТА

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.2
Р83

Рудикова Л. В.

Р83 Базы данных. Разработка приложений. — СПб.: БХВ-Петербург, 2006. — 496 с.: ил.

ISBN 5-94157-805-9

Книга является практическим руководством по созданию баз данных и приложений, использующих базы данных. Материал тщательно подобран с целью максимального удовлетворения запросов студенческой аудитории при сохранении компактного объема. Рассматриваются: реляционная модель данных, реляционная алгебра, язык SQL, создание пользовательских приложений средствами Microsoft Access, разработка клиент-серверных приложений с использованием InterBase и Delphi, новые направления в развитии баз данных и т. д. В книге более 110-ти разобранных примеров с пошаговыми инструкциями по их выполнению и свыше 230-ти задач для самостоятельного решения.

Для широкого круга пользователей и разработчиков баз данных

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.03.06.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 31.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-805-9

Оглавление

Введение **1**

Глава 1. Основы баз данных **3**

Понятие базы данных 5
Трехуровневая архитектура СУБД 9
Архитектура типичной СУБД 13
Обзор направлений, лежащих в основе современных СУБД 19
Задания 29

Глава 2. Реляционная модель данных **33**

Основные понятия реляционной модели данных 34
Структура данных реляционной модели 37
Структурная часть базы данных. Виды отношений 44
Реляционная целостность данных 46
Индексирование 48
Задания 52

Глава 3. Реляционная алгебра **55**

Основные определения, относящиеся к реляционной алгебре 55
Замкнутость реляционной алгебры 56
Отношения, совместимые по типу 57
Оператор переименования атрибутов 58
Традиционные операции над множествами (теоретико-множественные операторы) 59
Объединение 59
Пересечение 61
Вычитание 62
Декартово произведение 62

Специальные реляционные операторы 65
Выборка (ограничение, селекция) 65
Проекция 66
Соединение 67
Общая операция соединения 68
Тэта-соединение (Θ-join) 68
Экви-соединение 71
Естественное соединение (natural-join) 73
Деление 74
Примеры использования реляционных операторов 77
Внешние соединения 78
Задания 79

Глава 4. Основы языка SQL 83

Стандарт ANSI для языка SQL 83
Типы команд SQL 84
Сеанс SQL 87
Инструкции SQL 88
Типы данных 89
Домены 94
Константы 95
Выражения 96
Функции для работы со строками 97
Математические функции 102
Функции преобразования 102
Функции для работы с датами 103
Создание баз данных. Язык DDL 103
Схемы в SQL 104
Таблицы (отношения) 106
Создание таблицы 106
Определение столбца 106
Определение первичного и внешнего ключей 108
Условия уникальности 109
Условия на значения 109
Механизм проверки ограничений 110
Удаление таблицы 114
Изменение определения таблицы (*ALTER TABLE*) 114
Утверждения 116
Псевдонимы таблиц 117
Индексы 117

Представления ... 118
Другие объекты базы данных ... 121
Системный каталог ... 121
Манипуляция данными. Язык DML ... 123
Добавление новых данных ... 123
Удаление данных ... 124
Обновление данных ... 125
Запросы на выборку данных. Язык DQL ... 127
Инструкция *SELECT* для выборки данных ... 127
Предложение *SELECT* ... 127
Предложение *FROM* ... 128
Предложение *WHERE* ... 128
Статистические функции ... 134
Предложение *GROUP BY* ... 136
Предложение *HAVING* ... 137
Предложение *ORDER BY* ... 137
Объединения в многотабличных запросах на выборку ... 138
Объединение результатов нескольких запросов (операция *UNION*) ... 138
Объединение по равенству ... 139
Объединение по неравенству ... 140
Рекурсивное объединение (самообъединение) ... 140
Внутреннее объединение (*INNER JOIN*) ... 140
Перекрестное объединение (*CROSS JOIN*) ... 140
Полное внешнее объединение (*FULL JOIN*) ... 141
Левое внешнее объединение (*LEFT JION*) ... 141
Правовое внешнее объединение (*RIGHT JION*) ... 141
Расширенный запрос на объединение (*UNION JOIN*) ... 142
Задание объединений ... 142
Правила выполнения запроса на выборку ... 143
Некоторые замечания о подчиненных запросах ... 144
Примеры ... 146
База данных "Студенты" ... 146
База данных "Продажа товаров" ... 149
Задания ... 159

Глава 5. Вспомогательные аспекты баз данных ... 165

Целостность баз данных ... 165
Триггеры ... 168

Создание генераторов ... 171
Хранимые процедуры ... 172
Функции ... 186
Восстановление базы данных ... 189
Транзакции ... 189
Управление транзакциями ... 191
Журнал транзакций ... 193
Восстановление системы ... 193
Отказы системы ... 194
Отказы носителей ... 195
Параллелизм в базах данных ... 196
Проблемы параллелизма. Транзакции в многопользовательском режиме ... 196
Блокировка ... 197
Уровни блокировки ... 199
Допустимые комбинации блокировок для двух параллельно выполняемых транзакций ... 200
Тупиковые ситуации ... 201
Усовершенствованные методы блокировки ... 202
Явная блокировка ... 202
Уровни изоляции ... 202
Параметры блокировки ... 204
Интервал блокировки ... 205
Упорядоченность транзакций ... 205
Администрирование баз данных ... 207
Защита базы данных ... 207
Некомпьютерные средства контроля ... 211
Безопасность ... 212
Избирательное управление доступом ... 213
Контрольный след ... 216
Обязательное управление доступом ... 217
Поддержка мер обеспечения безопасности в языке SQL ... 217
Задания ... 221

Глава 6. Создание приложений средствами Microsoft Access ... 227

Общие замечания по созданию баз данных средствами Microsoft Access ... 227
Особенности интерфейса Microsoft Access ... 230

Создание базы данных ... 237
Создание новой базы данных ... 237
Создание базы данных на основе шаблонов ... 239
Создание таблиц и схемы данных ... 243
Общие рекомендации по созданию таблиц и схемы данных ... 243
Создание таблицы в режиме конструктора ... 245
Использование маски ввода ... 254
Выбор первичного ключа ... 256
Индексирование таблицы ... 257
Создание схемы данных ... 259
Изменение свойств полей и связей между таблицами ... 262
Ввод и редактирование данных в таблицах ... 262
Использование выражений ... 264
Обработка данных средствами Microsoft Access ... 280
Сортировка, поиск и фильтрация данных ... 280
Запросы в Microsoft Access ... 283
Общие сведения о запросах в Microsoft Access ... 283
Рекомендации по созданию запросов в Microsoft Access ... 286
Примеры запросов ... 297
Создание форм и отчетов. Использование макросов ... 321
Создание форм ... 321
Создание отчетов ... 335
Некоторые сведения о макросах ... 344
Придание приложению Microsoft Access законченного вида ... 350
Задания для самостоятельной работы ... 352
БД "Доставка товара" ... 352
БД "Туристическое агентство" ... 353
БД "Кинокомпания" ... 358

Глава 7. Создание клиент-серверных приложений средствами InterBase и Delphi ... 369

Принципы создания клиент-серверных приложений ... 369
Двухзвенная архитектура "клиент-сервер" ... 370
Трехзвенная архитектура "клиент-сервер" ... 373
Основные возможности сервера баз данных InterBase ... 377
Утилита IBConsole (InterBase Console) ... 377

Соединение с сервером ... 378
Создание базы данных ... 380
Соединение с базой данных ... 383
Выбор текущего сервера и базы данных ... 384
Разрыв соединения ... 385
Изменение свойств базы данных ... 385
Статистические данные о базе данных ... 386
Сборка мусора ... 391
Создание резервной копии (сохранение) и восстановление базы данных ... 392
Переход в однопользовательский режим соединения с базой данных ... 392
Резервное копирование базы данных ... 394
Восстановление базы данных из резервной копии ... 395
Принудительная запись на диск ... 398
Восстановление транзакций ... 398
Регистрация новых пользователей ... 399
Работа с утилитой BDE Administrator ... 400
Создание псевдонима базы данных ... 401
Создание псевдонима *INTRBASE* ... 402
Установки параметров драйвера ... 407
Системные стартовые установки ... 407
Установки форматов ... 408
Сохранение конфигурации ... 408
Пример разработки клиентского приложения с использованием InterBase и Delphi ... 410
Проектирование базы данных ... 410
Генерация SQL-скрипта ... 412
Создание базы данных с помощью утилиты IBConsole ... 423
Разработка приложения в среде Delphi ... 428
Мастер построения запросов ... 447
Листинг клиентского приложения ... 458
Главная форма ... 458
Мастер построения запросов ... 468
Информация о программе ... 475
Задания ... 476

Рекомендуемая литература ... 481

Предметный указатель ... 483

Введение

Организация автоматизированных систем занимает значительное место в становлении единого информационного пространства и компьютеризации многих аспектов промышленной, хозяйственной и учебной деятельности, различные направления которых связаны с хранением и обработкой больших массивов данных. Обращение к массивам данных лежит в основе практически любой компьютерной системы, построение которой в той или иной мере связано с использованием базы данных.

Книга является практическим руководством по реализации баз данных. Она написана с учетом современных требований, предъявляемых к правильной организации баз данных и охватывает обширный теоретический и практический материал: реляционная модель данных, реляционная алгебра, углубленное знакомство с языком SQL, создание пользовательских приложений средствами Microsoft Access, разработка клиент-серверных приложений средствами InterBase и т. д.

В книге использованы следующие программные продукты:

- Microsoft Access — система управления реляционными базами данных для создания различных автоматизированных приложений;
- InterBase — промышленный сервер баз данных для создания приложений типа "клиент-сервер";
- Delphi — объектно-ориентированная среда программирования для разработки клиентских приложений пользователя.

Представленные в книге примеры и задачи, а также большое количество индивидуальных заданий предназначены для

углубленного освоения различных практических аспектов, связанных с базами данных.

Книга состоит из 7 глав, каждая из которых посвящена определенной тематике реализации либо использования баз данных. Материал книги может быть полезен:

- в качестве учебного пособия для студентов, изучающих базы данных в различных курсах систем обработки данных, систем управления базами данных и т. д.;
- преподавателям при подготовке лекций и проведении практических и лабораторных работ;
- пользователям, разработчикам и программистам — для расширения профессиональных возможностей при использовании средств проектирования и реализации баз данных.

Автор выражает благодарность Зайцу Юрию Эдуардовичу, Горчичко Сергею Алексеевичу и Щегловой Ольге Леонидовне за помощь, оказанную при подготовке рукописи к изданию, а также всему издательскому коллективу группы "БХВ-Петербург".

Глава 1

Основы баз данных

В середине 60-х годов развитие вычислительной техники, а также потребности общества, прежде всего, потребности менеджмента, привели к появлению первых коммерческих информационных систем, которые позволили хранить длительное время и обрабатывать необходимую информацию, представляющую собой большие массивы данных.

Первые коммерческие информационные системы использовались, прежде всего, для ведения бухгалтерии: составление различных отчетов, ведомостей, сводов и т. д. Как правило, эти системы выполняли основные функции по обработке документов, в силу чего они получили название *систем обработки данных*.

С точки зрения организации данных это были файловые системы, позволявшие хранить большое количество информации в течение продолжительного времени. Однако первые информационные системы выполняли в основном лишь канцелярскую работу и обладали рядом существенных недостатков. В общем случае эти системы не давали гарантии, что данные не будут потеряны, если они не скопированы. Они не поддерживали также эффективного доступа к данным, расположенным в неизвестном файле, язык запросов на данные в файлах и т. д. Допуская параллельный доступ к файлам множества пользователей или процессов, они не предотвращают

ситуации изменения одного и того же файла одновременно многими пользователями, поэтому изменения одного из пользователей в файле могут вообще не появиться.

Более поздние системы перешли к накоплению и управлению информацией, которая на сегодняшний момент является важнейшим фактором существования любой организации.

Первые системы управления базами данных (СУБД) возникли из систем, которые обеспечивали пользователю возможность визуально воспринимать данные в основном так, как они хранились. В этих системах применялись различные модели данных для описания структуры хранимой информации в базе данных. Главные из них — иерархическая модель, основанная на деревьях, и сетевая модель, основанная на графах.

Недостаток первых моделей и систем состоял в том, что они не поддерживали языки запросов высокого уровня.

В 1970 году появилась статья Эдгара Кодда о представлении данных, организованных в виде двумерных таблиц, называемых отношениями (Codd E. F., "A relational model for large shared data banks", Comm. ACM, 13:6, pp. 377—387). С этого момента реляционная модель широко используется при создании различных баз данных. Следует отметить, что пользователь реляционной системы не связан со структурой памяти, в отличие от пользователя прежних систем БД. Запросы можно выражать на языке очень высокого уровня. Поставщиком первых реляционных и репрезентационных СУБД была фирма IBM.

Примерами систем, в которых используются базы данных, могут быть различные банковские системы, системы резервирования авиационных либо железнодорожных билетов, различные системы автоматизированного управления предприятием и т. д.

В настоящее время системы управления базами данных должны служить, прежде всего, основой информационных систем корпоративного управления и поддерживать принцип открытой системы, основу которого составляет согласованность

объединенного вместе различного оборудования и программного обеспечения.

Понятие базы данных

Под *системой с базой данных* обычно понимается любая информационная система на базе компьютера, в которой данные могут совместно использоваться многими приложениями.

Поясним также некоторые основные понятия, связанные с данным определением.

Данные — разрозненные факты предметной области (описания, опросные и анкетные данные, ведомости и т. д.).

Информация — организованные и обработанные данные, как правило, информация представляет собой структурированные факты предметной области.

Замечание

Часто при рассмотрении различных вопросов, связанных с обработкой информации, понятия "данные" и "информация" используются как синонимы. При этом имеется в виду, что обработка касается всегда только структурированных фактов предметной области, т. е. реально подразумевается обработка информации.

Информационная система — некоторая автоматизированная система на базе компьютера либо на базе иной вычислительной техники, которая организует данные и выдает требуемую информацию.

Информационно-управляющая система — некоторая автоматизированная система на базе компьютера либо на базе иной вычислительной техники, обеспечивающая информационную поддержку менеджмента.

Под *базой данных* понимается множество взаимосвязанных элементарных групп данных (информации), которые могут обрабатываться одной или несколькими прикладными системами.

Каждая СУБД должна удовлетворять следующим требованиям:

- обеспечивать пользователю возможность создавать новые базы данных и определять их *схему (логическую структуру данных)* с помощью специального языка — *языка определения данных*; поддерживать разнообразные представления одних и тех же данных;
- позволять *"запрашивать"* данные (информацию из базы) и изменять их с помощью *языка запросов*, или *языка манипулирования данными*; допускать интеграцию и совместное использование данных различными приложениями;
- поддерживать хранение очень больших массивов данных, измеряемых гигабайтами и более, в течение длительного времени, защищая их от случайной порчи и неавторизованного использования, а также обеспечивать модификацию базы данных в случае необходимости и доступ к данным путем запросов, т. е. гарантировать безопасность и целостность данных;
- контролировать доступ к данным одновременно для многих пользователей; исключать влияние запроса одного пользователя на запрос другого и не допускать одновременный доступ, который может испортить данные, т. е. гарантировать управление параллельным доступом к данным.

Таким образом, в системе с базой данных можно выделить несколько компонентов.

- Пользователи — люди, которые используют информацию, находящуюся в базе данных. Здесь можно выделить следующие группы пользователей:
  - системные администраторы — отвечают за основные операции системы;
  - администраторы базы данных — управляют работой СУБД и обеспечивают функционирование базы данных;
  - проектировщики базы данных — разрабатывают структуру базы данных;

- системные аналитики — определяют основные функции системы базы данных и проектируют формы ввода данных, отчеты и процедуры, с помощью которых обеспечиваются доступ к данным и манипулирование (добавление, изменение, удаление) данными;
- программисты — создают программный код;
- непосредственные пользователи — используют прикладные программы для выполнения необходимых операций по автоматизации деятельности некоторого подразделения и т. д.

❑ Приложения — программы пользователей, которым необходима информация из системы.

❑ СУБД — программное обеспечение, которое управляет доступом к данным и обеспечивает указанные функциональные возможности системы с базой данных.

❑ Информация — обработанные данные (строки, хранящиеся в файлах).

❑ Хост-система — компьютерная система, в которой хранятся файлы. Доступ к строкам данных осуществляется хост-системой. Роль СУБД состоит в том, чтобы генерировать запросы, позволяющие использовать функциональные возможности системы управления файлами хост-системы для обслуживания различных приложений. СУБД представляет собой дополнительный уровень программного обеспечения, надстроенный над программным обеспечением хост-системы.

❑ Оборудование — все системные программные средства (универсальный компьютер (mainframe), персональный компьютер (ПК), ноутбук, карманный компьютер).

❑ Периферийное оборудование — физические устройства, обеспечивающие ввод/вывод, а также электронные устройства для подключения дополнительных компьютеров и организации сети.

Графически систему с базой данных можно представить в виде логической последовательности уровней, представленной на рис. 1.1.

Рис. 1.1. Уровни системы с базой данных

На самом нижнем уровне находятся данные, хранящиеся в физических файлах (физическая память базы данных). На верхнем уровне располагаются приложения, у которых имеется собственное представление одних и тех же физических данных. Каждое представление базы данных предполагает определенную логическую структуру, построенную из лежащих в основе физических данных. Чтобы обеспечить интерфейс между физической памятью базы данных и ее разнообразными логическими версиями (множеством поддерживаемых представлений), СУБД, в свою очередь, также состоит из нескольких уровней.

Трехуровневая архитектура СУБД

Первые попытки стандартизации общей архитектуры СУБД относятся к 1971 году, в котором предложен двухуровневый подход к архитектуре СУБД на основе использования системного представления, включающего понятие схемы базы данных и пользовательских представлений, т. е. подсхем.

В 1978 году комитетом ANSI/SPARC (ANSI, American National Standard Institute — Национальный институт стандартизации США; SPARC, Standart Planning and Requirements Committee — Комитет планирования стандартов и норм) официально зафиксировано различие между логическим и физическим представлением данных. В частности, была предложена обобщенная структура систем с базой данных. Эта структура получила название *трехуровневой архитектуры*, включающей в себя внутренний, концептуальный и внешний уровни (рис. 1.2).

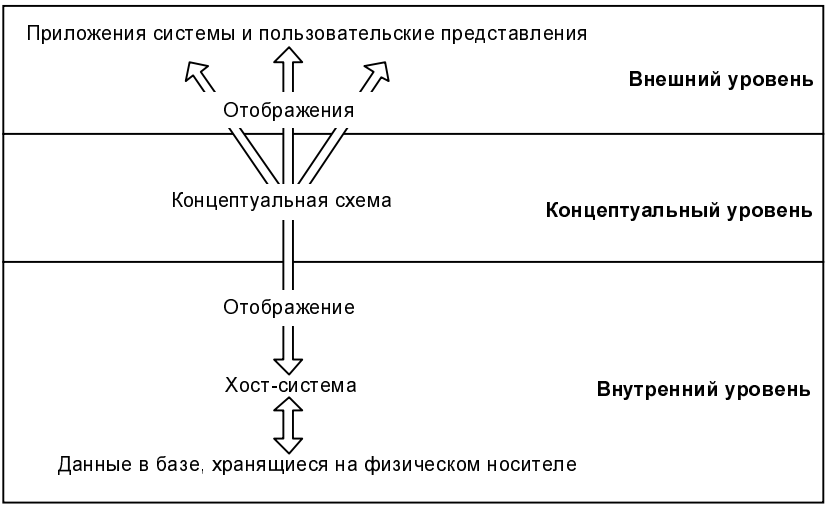


Рис. 1.2. Трехуровневая архитектура системы управления базой данных

Введение трехуровневой архитектуры базы данных позволило отделить пользовательское представление базы данных от ее физического представления. Оно обусловлено, прежде всего, следующими причинами:

- каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя собственное представление об этих данных, а также изменять его при необходимости, что не должно оказывать влияние на представления о данных других пользователей;
- обращение пользователя к базе данных не должно зависеть от особенностей хранения в ней данных;
- администратор базы данных может при необходимости изменять структуру хранения данных в базе, включая концептуальную структуру базы данных, причем данные действия не должны влиять на пользовательские представления данных;
- внутренняя структура хранения данных не должна зависеть от изменения физических устройств хранения информации.

Сформулируем различия между соответствующими уровнями архитектуры базы данных.

Внутренний уровень — уровень, определяющий физический вид базы данных, наиболее близкий к физическому хранению. Он связан со способами сохранения информации на физических устройствах. К нему имеют отношение дисководы, физические адреса, индексы, указатели и т. д. За этот уровень отвечают проектировщики физической базы данных, которые решают, какие физические устройства будут хранить данные, какие методы доступа будут использоваться для извлечения и обновления данных и какие меры следует принять для поддержания или повышения быстродействия системы управления базами данных. Пользователи не касаются данного уровня.

Концептуальный уровень — структурный уровень, который дает представление о логической схеме базы данных. На данном уровне выполняется концептуальное проектирование ба-

зы данных, которое включает анализ информационных потребностей пользователей и определение нужных им элементов данных. Результатом концептуального проектирования является концептуальная схема, логическое описание всех элементов данных и отношений между ними.

Внешний уровень — структурный уровень базы данных, определяющий пользовательские представления данных. Каждая пользовательская группа (либо пользователь) получает свое собственное представление данных в базе данных. Каждое такое представление данных дает ориентированное на пользователя описание элементов данных, из которых состоит представление данных и отношений между ними. Его можно напрямую вывести из концептуальной схемы. Совокупность таких пользовательских представлений данных и образует внешний уровень.

Под *схемой базы данных* понимается общее описание базы данных. В соответствии с трехуровневой архитектурой различают три различных типа схем базы данных.

Внешнему уровню представления базы данных соответствует, как правило, несколько *внешних схем (подсхем)*, которые соответствуют различным представлениям данных пользователей СУБД.

Концептуальная схема описывает все элементы данных, связи между ними, а также необходимые ограничения для поддержки целостности данных. Для каждой базы данных имеется только одна концептуальная схема данных.

Внутренняя схема является полным описанием внутренней модели данных и содержит определения хранимых записей, методы представления, описания полей данных, сведения об индексах и выбранных схемах хеширования. По аналогии с концептуальной схемой, для каждой базы данных имеется также только одна внутренняя схема.

Исходя из трехуровневого представления, а также из различных схем базы данных, следует, что СУБД должна устанавливать соответствие и следить за непротиворечивостью типов схем: внешними, концептуальной и внутренней. Концепту-

альная схема является центральным связующим звеном между каждой внешней схемой и внутренней схемой базы данных. Концептуальная схема связана с внутренней схемой посредством внутреннего концептуального отображения, в свою очередь, каждая внешняя схема связана с концептуальной схемой с помощью внешнего концептуального отображения, которое позволяет отображать пользовательское представление на соответствующую часть концептуальной схемы.

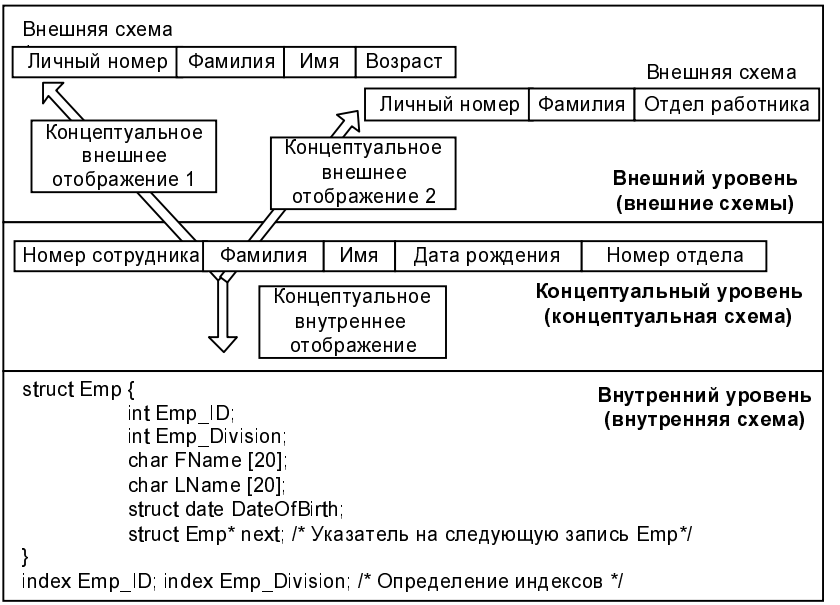


Рис. 1.3. Различия в схемах базы данных

Так (рис. 1.3), `Внешняя схема 1` предоставляет следующую информацию о сотрудниках учреждения: личный номер, фамилия, имя, возраст; `Внешняя схема 2`: личный номер, фамилия, отдел работника. Данные схемы получены на основе единой концептуальной схемы базы данных, причем поле `Возраст` получается путем вычисления на основе поля `Дата рождения`. В свою очередь, концептуальная схема преобразует-

ся в СУБД с помощью концептуального внутреннего отображения во внутреннюю схему, содержащую физическое описание структуры записи концептуального представления.

В теории и практике баз данных следует различать также описание базы данных и саму базу данных. Под *описанием базы данных* понимается схема базы данных, которая создается в процессе проектирования базы данных, изменение которой предполагается в исключительных случаях. Понятие *базы данных* предполагает всю информацию, содержащуюся в базе, которая может изменяться с течением времени. Совокупность информации, хранящейся в базе данных в любой определенный момент времени, называется *состоянием базы данных*. Таким образом, одной и той же схеме базы данных может соответствовать множество различных состояний базы данных.

Часто встречаются также следующие понятия для схемы и состояния базы данных. Схема базы данных называется *содержанием* базы данных, а ее состояние — *детализацией*.

Главное назначение трехуровневой архитектуры — обеспечение *независимости от данных*, т. е. любые изменения на нижних уровнях базы данных не должны влиять на верхние уровни. Независимость от данных бывает двух типов:

- логическая — полная защищенность внешних схем от изменений, которые вносятся в концептуальную схему;
- физическая — защищенность концептуальной схемы от изменений, которые вносятся во внутреннюю схему.

Архитектура типичной СУБД

На рис. 1.4 представлены главные компоненты архитектуры типичной СУБД. Рассмотрим назначение каждого компонента.

Компонент `Данные, Метаданные` включает не только данные, но также информацию о структуре данных *(метаданные)*. В реляционной СУБД метаданные включают в себя системные таблицы (отношения), имена отношений, имена атрибутов этих отношений и типы данных этих атрибутов. Как пра-

вило, СУБД поддерживает индексы данных. *Индекс* — это структура данных, которая помогает быстро найти элементы данных при наличии части их значения. Индексы представляют собой часть хранимых данных, а описания, указывающие, какие данные имеют индексы, — часть метаданных.

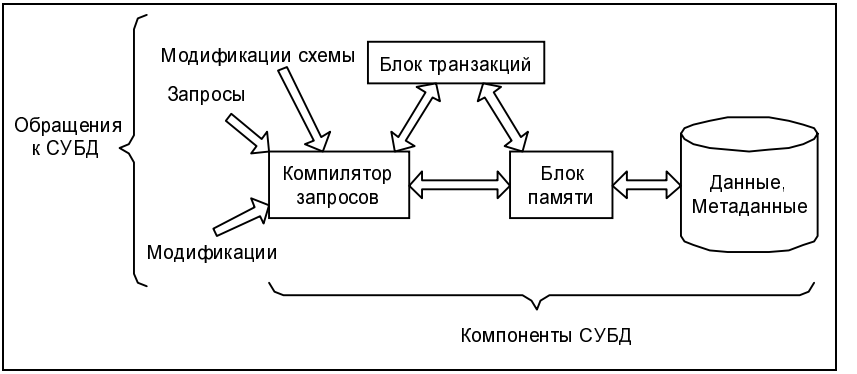


Рис. 1.4. Главные компоненты СУБД

Компонент `Блок памяти` получает требуемую информацию из места хранения данных и изменяет в нем соответствующую информацию по требованию расположенных выше уровней системы.

В простых системах баз данных блоком памяти может служить система файлов операционной системы. Однако для повышения эффективности СУБД обычно осуществляет прямой контроль памяти. Блок памяти состоит из двух компонентов.

- *Блок файлов* контролирует расположение файлов на диске и получает блок или блоки, содержащие файлы, по запросу блока буфера (диск в общем случае делится на *дисковые блоки* — смежные области памяти, содержащие от 4000 до 16 000 байт).
- *Блок буфера* управляет основной памятью. Он получает блоки данных с диска через блок файлов и выбирает стра-

ницу основной памяти для хранения конкретного блока. Блок буфера может временно сохранять дисковый блок в основной памяти, но возвращает его на диск, когда страница основной памяти нужна для другого блока. Страницы также возвращаются на диск по требованию блока транзакций.

Компонент `Компилятор запросов` — обрабатывает различные обращения к СУБД (запросы) и запрашивает изменения данных или метаданных. Он предлагает лучший способ выполнения необходимой операции и выдает соответствующие команды блоку памяти.

Компилятор запросов преобразует запрос или действие с базой данных, которые могут быть выполнены на очень высоком уровне (например, в виде запроса SQL), в последовательность более простых запросов. Часто самой трудной частью обработки запроса является его *организация*, т. е. выбор хорошего *плана запроса* или последовательности запросов к системе памяти, отвечающей на запрос.

Как правило, компилятором запросов обрабатываются три типа обращений к СУБД.

- *Запросы* — вопросы, касающиеся данных, находящихся в базе. Запросы могут генерироваться двумя способами:
  - с помощью общего интерфейса запросов (например, запросы, сформулированные на языке запросов высокого уровня — SQL, которые передаются компилятору запросов, он также получает ответы на них; как правило, реляционная СУБД поддерживает SQL-запросы);
  - с помощью интерфейсов прикладных программ (запросы передаются через специальный интерфейс, который предполагает генерацию только заданных запросов к базе, например, посредством полей ввода, полей со списком и т. д.; через данный интерфейс нельзя передавать произвольные запросы).
- *Модификации (модифицирующие запросы)* — операции по изменению данных (удаление, изменение, добавление). Они

также могут выполняться либо с помощью общего интерфейса, либо через интерфейс прикладной программы.

- *Модификации схемы базы данных* — это команды администраторов базы данных, которые имеют право изменять схему базы данных либо создавать новую базу данных.

Компонент `Блок транзакций` отвечает за целостность системы и должен обеспечить одновременную обработку многих запросов, отсутствие интерференции запросов (интерференция — сложение, в данном случае необходимо исключить наложение запросов и их взаимовлияние и т. д.) и защиту данных в случае выхода системы из строя. Блок транзакций взаимодействует с компилятором запросов, т. к. для разрешения конфликтных ситуаций должен учитывать, на какие данные воздействуют текущие запросы. В силу этого некоторые запросы могут быть отложены и установлена очередность их выполнения. Блок транзакций взаимодействует также с блоком памяти, т. к. схемы защиты данных обычно включают в себя хранение файла регистрации изменений данных. При правильном порядке выполнения операции файл *регистрации* содержит запись изменений, поэтому можно заново выполнить даже те изменения в базе данных, которые из-за сбоя в системе были прерваны и не внесены в физическое пространство диска.

Типичные СУБД позволяют пользователю выполнить несколько запросов и/или изменений в одной транзакции. Под *транзакцией* понимается совокупность действий (группа операций), которые необходимо выполнить последовательно, но которые будут восприниматься как единое целое.

Как правило, система базы данных поддерживает одновременно множество транзакций. Именно правильное выполнение всех таких транзакций и обеспечивает на схеме (рис. 1.4) `Блок транзакций`. Правильное выполнение транзакций обеспечивается ACID-свойствами (Atomicity, Consistency, Isolation, Durability):

- *атомарность* — требование выполнения либо всех транзакций, либо ни одной из них (например, изъятие денег со счета в банке данного клиента и внесение соответствую-

щих изменений должны быть единственной атомарной транзакцией, не допускающей выполнение каждой из этих операций по отдельности);

- *непротиворечивость* — состояние, при котором данные соответствуют всем возможным ожиданиям (например, условие непротиворечивости для базы данных бронирования авиационных билетов состоит в том, что ни одно из мест в самолете не бронируется для двух пассажиров);
- *изоляция* — предполагает, что при параллельном выполнении двух или более транзакций их результаты должны быть изолированы друг от друга. Одновременное выполнение двух транзакций не должно приводить к результату, которого не было, если бы они выполнялись последовательно (например, при продаже авиационных билетов на один и тот же рейс в случае последнего свободного места при одновременном запросе-обращении к базе двух служащих запрос одного должен быть выполнен, другого — нет);
- *долговременность* — предполагает, что после завершения транзакции результат не должен быть утрачен при сбоях в системе (даже если сбой происходит сразу после окончания транзакции).

Следует также отметить, что современное программное обеспечение поддерживает архитектуру *"клиент-сервер"*. Данная концепция предполагает, что один процесс (*клиент*) посылает запрос для выполнения другому процессу (*серверу*). Как правило, база данных часто разделяется на процесс сервера и несколько процессов клиента (рис. 1.5).

В простейшей архитектуре "клиент-сервер" вся СУБД является сервером, за исключением интерфейсов запроса, которые взаимодействуют с пользователем и посылают запросы или другие команды на сервер. Так, реляционная СУБД часто использует язык SQL для представления запросов от клиента к серверу. Затем сервер базы данных предоставляет клиенту ответ в виде соответствующей таблицы, в которой отображает-

ся интересующая информация. Существует тенденция увеличения нагрузки на клиентах, т. к. при наличии множества одновременно работающих пользователей базы данных могут возникнуть проблемы с сервером.

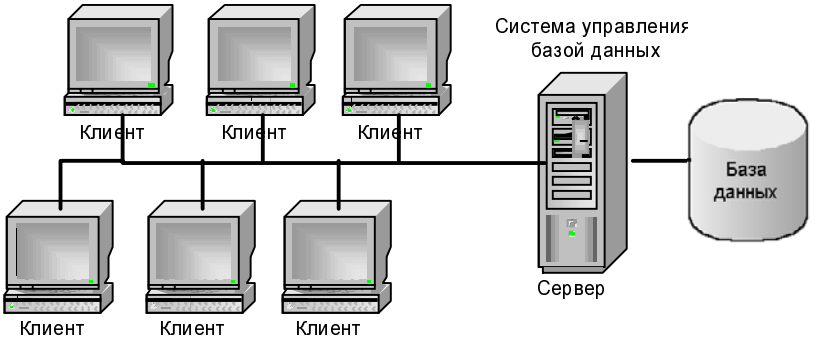


Рис. 1.5. Главные компоненты архитектуры клиент-сервер

В настоящее время СУБД получили широкое распространение в различных сферах деятельности и выполняют основную часть процедур, связанных с накоплением и обработкой информации. В связи с этим в табл. 1.1 приведем классификацию типов СУБД, зависящую от количества пользователей, местоположения базы данных и сферы использования.

Таблица 1.1. *Типы систем управления базами данных*

Способ классификации	Тип СУБД	Ключевые признаки	
По количеству пользователей	Однопользовательская	В конкретный момент времени с базой данных работает только один пользователь	
		Настольная база данных	На базе персонального компьютера

Таблица 1.1 *(окончание)*

Способ классификации	Тип СУБД	Ключевые признаки	
	Много-пользова-тельская	База данных рабочей группы	Число пользова-телей менее 50 человек
		База данных предприятия	Большое число пользователей (более 50 человек)
По месту разме-щения базы данных	Централи-зованная	СУБД работает с базой данных, размещенной на одном сервере	
	Распреде-ленная	СУБД работает с базой данных, размещенной на нескольких серверах	
По способу при-менения и сфере использования	Транзак-ционная (рабочая или опера-ционная)	СУБД работает с базой данных, в которой для транзакций отводит-ся минимальное время и результа-ты запросов к базе должны отобра-жаться в наикратчайшие сроки	
	База дан-ных под-держки ре-шений (хранилище данных)	СУБД работает с базой данных, предназначенной для получения необходимой информации при вы-работке стратегических или такти-ческих решений с целью оптимиза-ции деятельности предприятия	

Обзор направлений, лежащих в основе современных СУБД

Теория и практика баз данных в настоящее время развиваются в различных направлениях. Это, прежде всего, новые технологии, объектно-ориентированное программирование, ограничения и триггеры, мультимедийные данные, Интернет, новые приложения типа хранилища или интеграции данных.

Архитектура "клиент-сервер" составляет основу достаточно широкого класса современного программного обеспечения. В системах с данной архитектурой могут происходить два типа независимых процессов: процесс клиента и процесс сервера, которые могут выполняться как на одном компьютере, так и на различных компьютерах, подключенных к сети. В общей модели "клиент-сервер" предполагается четкое разделение процессов клиента и сервера, которые не зависят друг от друга. Серверы и клиенты могут находиться друг с другом в связи "многие-ко-многим": один сервер может предоставлять сервисы для многих клиентов, в свою очередь, каждый клиент может обращаться с запросами на многие серверы.

Объектно-ориентированное программирование предлагает для систем баз данных следующие возможности:

- оперирование данными с помощью развитой системы типов данных в более естественных формах, чем в классических моделях баз данных;
- с помощью классов и иерархии классов возможно более простое совместное либо повторное использование программного обеспечения;
- использование абстрактных типов данных предотвращает неправильное использование данных (в том случае, если разрешить доступ к ним только через точно спроектированные функции, использующие указанные данные правильно).

Ограничения и триггеры представляют собой активные элементы базы данных, которые выполняют указанные в них действия в подходящие моменты. Такие элементы особенно актуальны в различных коммерческих системах.

Ограничения — булевы функции, значения которых должны быть истинными. Например, в базу данных бухгалтерии вводится некоторое ограничение, запрещающее отрицательный баланс счета. Любое изменение в такой базе данных, нарушающее данное ограничение и приводящее к отрицательному балансу, отвергается СУБД.

Триггеры — части программы, реагирующие на определенное событие. Событиями могут быть различные действия: добав-

ление, удаление либо изменение элементов данных определенного типа. Если происходит данное событие, то выполняется заданная последовательность действий. Например, если изменяется должность работающего в организации, то происходит автоматический перерасчет базовых начислений на зарплату данного сотрудника.

С активными элементами в более ранних СУБД возникали технические трудности из-за их неэффективной реализации в случае большого объема данных, на которые они воздействуют. Однако в современных СУБД данные активные элементы применяются достаточно широко.

Поддержка *данных мультимедиа*, т. е. данных большого объема, способных изменяться в широких пределах. Это, например, такие типы данных, как аудио- и видеозаписи, сигналы, полученные от радаров, спутниковые изображения, графика различных форматов и т. д. Для хранения подобных данных создаются различные средства расширения СУБД. Операции, выполняемые с данными мультимедиа, не являются простыми и не подходят для традиционных форм данных. СУБД должна включать в себя возможность ввода пользователями по собственному выбору функций, применимых к данным мультимедиа. Часто в системах, использующих данные мультимедиа, применяются объектно-ориентированные методы.

Интеграция данных предполагает получение необходимой информации из различных баз и источников данных, а также ее обработку. Концепция *хранилищ данных*, в которые копируется информация из множества наследственных баз данных и соответствующим образом переводится в центральную базу данных, а также система оперативного анализа данных OLAP (On-Line Analytical Processing) используются в настоящее время в системах поддержки решений (DSS). Это несомненно важно для: планирования и анализа производства, оперативного складского учета, изучения информационных потребностей в торговой сфере, в сфере рекламного бизнеса различных уровней, в электронных каталогах распространения через Интернет и т. д.

Наиболее важные модели, технологии и системы управления базами данных приведены в табл. 1.2.

Таблица 1.2. *Хронология развития систем управления базами данных*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
Конец 50-х гг.	Файловые системы	Файлы последовательного доступа	Обработка записей информации производилась в последовательном порядке	Первые системы бронирования авиабилетов; банковские системы; корпоративные системы и др.
1960 г.		Файлы произвольного доступа	Поддержка прямого доступа к конкретной записи информации	
1960—1965 гг.	Иерархические системы управления (иерархическая или древовидная модель)	Поддержка доступа к нескольким записям информации	Отношения между данными подчиняются определенной иерархии, основу которой составляет отношение "потомок-предок", зависящее от предопределенных физических указателей	Проект компании North American Rockwell, поддерживавшей высадку астронавтов на Луне (1968 г.). Система учета банковских счетов
1965—1970 гг.	Сетевые системы управления (сетевая или графовая модель)	Поддержка иерархических и неиерархических отношений между данными	Сетевая модель похожа на иерархическую, однако записи сетевой модели могут иметь более одного предка	Базы данных торговых предприятий

Таблица 1.2 *(продолжение)*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
1970 г.	Реляционная модель (появление в печати статьи Кодда о реляционных отношениях)	Поддержка всех логических отношений между данными	Реляционная модель представляет собой набор взаимосвязанных таблиц, что обеспечивает простой доступ к данным	Появление в конце 70-х гг. XX века реляционных систем управления базами данных, которые расширили в дальнейшем возможности СУБД, повысили доступность информации и т. д. Практическое использование реляционных систем управления базами данных — сферы управления различных организаций для соответствующих нужд, сфера торговли, транспорта, маркетинг услуг и т. д.
1970—1990 гг. и далее	Реляционная модель			
1976 г.	ERM (Entity-Relationship Model, модель "сущность-связь"). Публикация статьи Питера Чена о графическом представлении логических объектов базы данных	Поддержка всех логических отношений между данными	Модель "сущность-связь" предлагает графическое представление логических объектов (сущностей) и их отношений (связей) в структуре базы данных	Простой графический инструмент моделирования данных для разработки реляционных баз данных на концептуальном уровне

Таблица 1.2 *(продолжение)*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
1981 г.	SDM (Semantic Database Model, семантическая модель БД), предложенна в статье М. Хаммера и Д. Маклеодома	SDM моделирует как данные, так и их отношения в единой структуре, которая называется объектом	OODM-модель отражает объекты базы данных, содержащих как внутреннюю семантику объекта, так и информацию о связях с другими объектами. Основа концепции OODM-модели — объектно-ориентированная модель и семантическая модель данных	Современные базы данных часто используют графику, видео и звук наряду с текстовой и цифровой информацией. Использование сложных типов данных, повторное использование кода, повышение требований к системе и т. д. стало выходить за рамки возможностей реляционного подхода. В силу этого объектно-ориентированная модель данных становится во многих случаях основой для создания необходимой информационной системы. OODM-модель используется в настоящее время при разработке специальных инженерных и научных приложений. Основы OODM-модели разработаны компанией Xerox, достижения которой в настоящее время известны благодаря графическому интерфейсу пользователя, компьютерной мыши, электронной почте, объектно-ориентированным языкам программирования

Таблица 1.2 *(продолжение)*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
Начиная с 80-х гг. и далее	OODM (Object Oriented Database Model, объектно-ориентированная модель	Поддержка логических отношений между данными, представленных в виде объектов с семантическим наполнением		
Начиная с 80-х гг. и далее	ERDM (Extendet Relation Data Model, расширенная реляционная модель), а также: O/RDM (Object/ Relational Data Model, объектно-реляционная модель)	Поддержка всех логических отношений между данными	ERDM-модель базируется на концепциях реляционной модели	ERDM-модель используется в основном для бизнес-приложений. Часто СУБД, основанные на ERDM, называют объектно-реляционными системами управления базой данных (ОРСУБД)
Начиная с 1990 г.	Client-server (технология "клиент-сервер")	Сочетание распределенной обработки данных с централизованным управлением и доступом к данным	Разделение функций СУБД между процессом сервера и одним либо несколькими процессами клиента. Простейший вариант архитектуры "клиент-сервер" предполагает, что СУБД целиком представляет собой сервер, за исключением	Системы, предполагающие наличие сети, а также необходимость централизованного хранения и управления данными. В настоящее время клиент-серверные приложения находят широкое применение в различных промышленных и социальных организациях и структурах

Таблица 1.2 *(продолжение)*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
			интерфейсов приложений конечного пользователя, откуда отсылаются запросы к данным и другие команды с целью их выполнения сервером	
Начиная с 2000 г.	Технология многоуровневой архитектуры	Сочетание распределенной обработки данных с централизованным управлением и доступом к данным, используя промежуточное звено — сервер приложений (application server)	СУБД выполняет роль сервера базы данных (поставка динамически генерируемого содержимого Web-сайта), клиентом которого служит сервер приложений (управляет подключениями к базе данных, транзакциями, авторизацией и другими процессами). В свою очередь, клиентами сервера приложений служат Web-серверы, предоставляющие различные услуги пользователям, а также другие программные приложения	Использование многоуровневой архитектуры особенно эффективно в последнее время в связи с развитием сети Интернет и предоставлением через нее различных услуг конечным пользователям

Таблица 1.2 *(продолжение)*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
Начиная с 2000 г.	Технологии интеграции информации	Федеративные базы данных (federated databases — независимые источники данных с возможностью получения необходимой информации из других источников данных)	Возможность использования содержимого двух либо более баз данных (источников информации — information source) с возможностью дальнейшего создания единой крупной базы данных (возможно, виртуальной — virtual database), к которой можно обращаться с запросами, как к единому виртуальному пространству	Системы, реализованные с применением технологий интеграции информации (OLAP-системы), встречаются в крупных промышленных и социальных организациях, а также в Интернете — там, где необходимы: оптимизация бизнес-процессов, аналитическая обработка информации большого объема, выявление и прогнозирование различных тенденций, поддержка принятия решений и т. д.
		Хранилища данных (data warehouses) — хранение копий фрагментов информации нескольких источников данных, прошедших определенное преобразование с целью согласования структур с общей схемой хранилища данных, в единой базе данных. Обновление хранилища происходит регулярно		

Таблица 1.2 *(окончание)*

Период появления	Название модели/технологии		Описание	Некоторые примеры и замечания
		Медиаторы (mediators) — программные компоненты, которые обеспечивают поддержку виртуальных баз данных с возможностью обработки запросов		

Задания

1. Дать определение следующих терминов:
   - данные;
   - информация;
   - информационная система;
   - информационно-управляющая система;
   - база данных;
   - система с базой данных;
   - система управления базой данных;
   - схема базы данных;
   - состояние базы данных;
   - содержание базы данных;
   - детализация;
   - метаданные;
   - индекс;
   - транзакция;
   - клиент;
   - сервер.
2. Какие из приведенных далее утверждений можно считать данными, а какие рассматривать как информацию?
   - Петров Иван Олегович родился 12 августа 1975 года;
   - товар Шайба-322 отправлен;
   - служащий Иванов Г. П. получил в данном квартале выручку от продаж, намного большую, чем другие служащие его отдела;

- зарплата Котова И. С. за июнь месяц 2005 года составила 6800 рублей;
- за последние месяцы текущего года наблюдался заметный рост продаж кондитерских изделий.

3. Каким требованиям должна удовлетворять система управления базой данных?
4. Какие основные компоненты можно выделить в системе с базой данных?
5. Объяснить основные функции следующих групп пользователей:
   - системный администратор;
   - администратор базы данных;
   - прикладной программист;
   - обычный пользователь;
   - проектировщик базы данных;
   - системный аналитик.
6. Определить основные преимущества использования системы с базой данных.
7. Определить недостатки использования системы с базой данных.
8. Каковы преимущества трехуровневой архитектуры системы управления базой данных? Сформулировать основные отличительные черты каждого уровня и определить главное назначение трехуровневой архитектуры.
9. Что понимается под логической независимостью от данных?
10. Что понимается под физической независимостью от данных?
11. Сколько различают типов схем базы данных и каково их назначение?
12. Дана следующая информация, касающаяся концептуальной схемы базы данных: `Номер_зачетной_книжки`,

Фамилия_Студента, Имя_Студента, Отчество_Студента, Дата_рождения, Дата_поступления, Факультет, Специальность, № Группы, Семейное_положение, Особые_отметки. Получить несколько внешних схем, отвечающих потребностям различных пользователей системы.

13. Определить главные компоненты СУБД и дать краткую характеристику назначения каждого компонента.
14. Какие типы обращений к базе данных обрабатываются СУБД?
15. Что понимается под ACID-свойствами? Привести соответствующие примеры ситуаций, которые могут вызвать конфликты в системе с базой данных.
16. Определить основные компоненты архитектуры "клиент-сервер". Привести примеры использования данной архитектуры.
17. Дать краткую характеристику следующим типам СУБД:
    - однопользовательская;
    - многопользовательская;
    - централизованная;
    - распределенная;
    - транзакционная;
    - хранилище данных.

Глава 2

Реляционная модель данных

Реляционная модель в настоящее время доминирует на рынке баз данных. Основу этой модели составляет набор взаимосвязанных таблиц, в которых хранятся данные.

Основные теоретические идеи реляционной модели были изложены в работах по теории отношений американского логика Чарльза Содерса Пирса (1839—1914) и немецкого логика Эрнста Шредера (1841—1902), а также американского математика Эдгара Кодда. В работах Пирса и Шредера было доказано, что множество отношений замкнуто относительно некоторых специальных операций, совместно образующих абстрактную алгебру. В дальнейшем это важнейшее свойство отношений было использовано в реляционной модели для разработки языка манипулирования данными. В 1970 году появилась статья Эдгара Кодда о представлении данных, организованных в виде двумерных таблиц, называемых отношениями (Codd E. F., "A relational model for large shared data banks", Comm. ACM, 13:6, pp. 377—387). В этой работе впервые введены основные понятия и ограничения реляционной модели как основы хранения данных, а также показана возможность обработки данных с помощью традиционных операций над множествами и специальных введенных реляционных операций.

Реляционная модель либо ее основы широко используются при создании различных баз данных. Представление данных в

виде множественной совокупности таблиц позволило избежать многих недостатков ранних СУБД и создать системы с упрощенным интерфейсным управлением. В настоящее время многие серверы баз данных основаны на принципах работы, связанных с реляционной моделью. В связи с этим следует подробно рассмотреть основные положения реляционной модели данных.

Основные понятия реляционной модели данных

Основные объекты реляционной модели данных перечислены в табл. 2.1.

Таблица 2.1. *Элементы реляционной модели*

Реляционный термин	**Описание**
База данных	Набор таблиц и других объектов, необходимых для абстрактного представления части реального мира (решаемой задачи)
Схема базы данных	Набор заголовков таблиц, взаимосвязанных друг с другом
Отношение	Таблица — совокупность объектов реального мира, которые характеризуются общими свойствами и характеристиками (поля таблицы)
Заголовок отношения	Заголовок таблицы — названия полей (столбцов) таблицы
Тело отношения	Тело таблицы — совокупность значений для всех объектов реального мира, которая представима в виде записей таблицы (строки таблицы)
Схема отношения	Строка заголовков столбцов таблицы (заголовок таблицы)
Атрибут отношения	Наименование столбца таблицы (поле таблицы)

Таблица 2.1 *(продолжение)*

Реляционный термин	Описание
Кортеж отношения	Строка таблицы (запись) — однозначное представление объекта реального мира, созданное с использованием значений полей таблицы
Домен	Множество допустимых значений атрибута
Значение атрибута	Значение поля в записи
Первичный ключ	Один или несколько атрибутов, который уникальным (единственным) образом определяет значение кортежа (значение строки таблицы)
Внешний ключ	Атрибут таблицы, значения которого соответствуют значениям первичного ключа в другой связанной таблице. Внешний ключ может состоять как из одного, так и из нескольких атрибутов (составной внешний ключ). Если число атрибутов внешнего ключа меньше, чем количество атрибутов соответствующего первичного ключа, то он называется усеченным (частичным) внешним ключом
Степень (арность) отношения	Количество столбцов таблицы
Мощность отношения	Количество строк таблицы (количество кортежей)
Тип данных	Тип значений элементов таблицы
Базовое отношение	Отношение, которое содержит один или несколько столбцов, характеризующих свойства объекта, а также первичный ключ
Производное отношение	Не является базовым отношением, т. е. не характеризует свойства объекта и используется для обеспечения связей между другими таблицами, может не содержать первичного ключа; если первичный ключ задан, то он состоит из внешних ключей, которые связаны с первичными ключами базового отношения

Таблица 2.1 *(окончание)*

Реляционный термин	Описание
Связь	Устанавливает взаимосвязь между совпадающими значениями в ключевых полях — первичным ключом одной таблицы и внешним ключом другой таблицы
Связь "один-к-одному"	При использовании такого вида связи запись в одной таблице может иметь не более одной связанной записи в другой таблице. Данная связь используется для разделения достаточно широких таблиц, либо по требованию защиты. В обеих таблицах ключевые поля должны быть первичными
Связь "один-ко-многим"	При использовании связи "один-ко-многим" каждой записи первой таблицы может соответствовать несколько записей второй таблицы, а каждой записи второй таблицы может соответствовать лишь одна запись первой таблицы. В первой таблице обязательно должен быть задан первичный ключ, во второй — внешний
Связь "многие-ко-многим"	При данном типе связи одной записи в первой таблице может соответствовать несколько записей второй таблицы и наоборот. Уникальность ключей для таких таблиц не требуется. Как правило, для разрешения таких связей необходимо ввести производное отношение и разрешить такую связь двумя связями "один-ко-многим", в результате чего появляется дополнительное производное отношение

В реляционной модели рассматриваются следующие аспекты, касающиеся данных:

- структура данных;
- целостность данных;
- обработка данных.

Объектами в основном являются таблицы (отношения), целостность обеспечивается внешними и первичными ключами, операторы — это набор инструкций, которые обеспечивают выборку и манипуляцию над данными (например, оператор `RESTRICT`, `JOIN` и т. д.).

Структура данных реляционной модели

Структура данных предполагает представление предметной области рассматриваемой задачи в виде набора взаимосвязанных отношений. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними. В каждой связи одно отношение может выступать как основное (базовое), а другое отношение выступает в роли подчиненного (производного). Таким образом, один кортеж основного отношения может быть связан с несколькими кортежами подчиненного отношения. Для поддержки этих связей оба отношения должны содержать наборы атрибутов, по которым они связаны. В основном отношении это первичный ключ отношения, который однозначно определяет кортеж основного отношения. В подчиненном отношении для моделирования связи должен присутствовать набор атрибутов, соответствующий первичному ключу основного отношения. Однако здесь этот набор атрибутов уже является вторичным ключом или внешним ключом, т. е. он определяет множество кортежей отношения, которые связаны с единственным кортежем основного отношения. Множество взаимосвязанных друг с другом таблиц образуют схему базы данных.

Рассмотрим основные элементы данных реляционной модели (рис. 2.1).

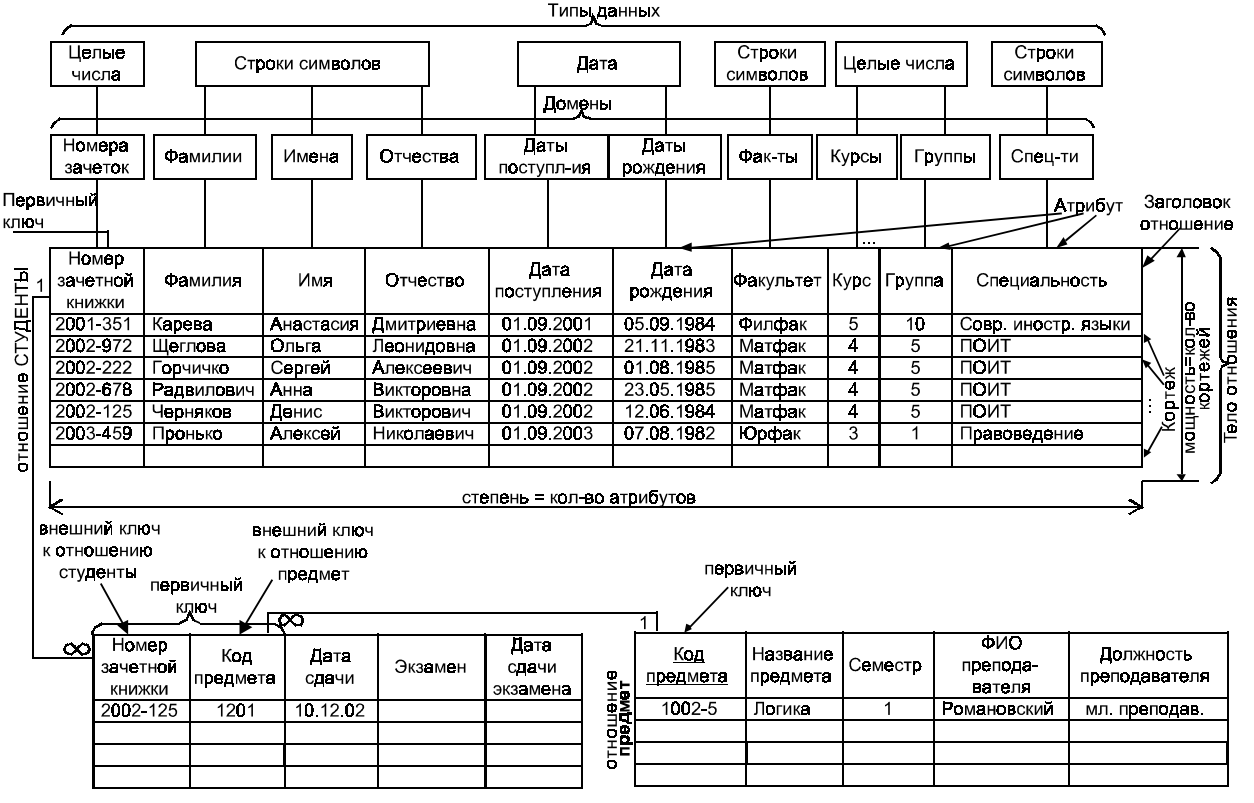


Рис. 2.1. Элементы реляционной модели данных

Отношение R представляет собой двумерную таблицу, содержащую некоторые данные. Математически N-арное отношение R — это множество декартова произведения $D_1 \times D_2 \times ... \times D_n$ множеств (доменов) D_1, D_2, …, D_n ($n \geq 1$), необязательно различных:

$$R \subseteq D_1 \times D_2 \times ... \times D_n ,$$

где $D_1 \times D_2 \times ... \times D_n$ — полное декартово произведение, т. е. набор всевозможных сочетаний из n элементов каждое, где каждый элемент берется из своего домена.

Домен представляет собой семантическое понятие, которое можно рассматривать как подмножество значений некоторого типа данных, имеющих определенный смысл. Домен характеризуется следующими свойствами:

- имеет уникальное имя (в пределах базы данных);
- определен на некотором простом типе данных или на другом домене;
- может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для этого домена;
- несет определенную смысловую нагрузку.

Замечание

Если тип данных можно считать множеством всех возможных значений текущего типа, то домен напоминает подмножество в этом множестве. Отличие домена от понятия подмножества состоит именно в том, что домен отражает семантику, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл. Например, домены `Вес товара` и `Количество товара` можно задать как множество неотрицательных целых чисел, однако смысл этих доменов будет различным, в силу чего они являются различными доменами. Основное значение доменов состоит в том, что они ограничивают сравнения: нельзя сравнивать значения из различных доменов, даже если они имеют одинаковый тип данных.

Атрибут отношения представляет собой пару вида <Имя_атрибута : Имя_домена> (либо $< A : D >$). Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Отношение R, определенное на множестве доменов, содержит две части: заголовок и тело.

Заголовок отношения — это фиксированное количество атрибутов отношения, описывающее декартово произведение доменов, на котором задано отношение:

$$(< A_1 : D_1 >, < A_2 : D_2 >, ..., < A_n : D_n >).$$

Заголовок статичен: он не меняется во время работы с базой данных. Если в отношении изменены, добавлены или удалены атрибуты, то в результате получается уже другое отношение (даже если его имя осталось прежним).

Тело отношения содержит множество кортежей отношения. Каждый кортеж отношения представляет собой множество пар вида <Имя_атрибута : Значение_атрибута>:

$$R(< A_1 : Val_1 >, < A_2 : Val_2 >, ..., < A_n : Val_n >)$$

таких, что значение Val_i атрибута A_i принадлежит домену D_i. Тело отношения представляет собой набор кортежей, т. е. подмножество декартового произведения доменов. Таким образом, тело отношения собственно и является отношением в математическом смысле слова. Тело отношения может изменяться во время работы с базой данных, т. к. кортежи с течением времени могут изменяться, добавляться и удаляться.

Отношение обычно записывается в виде:

$$R(< A_1 : D_1 >, < A_2 : D_2 >, ..., < A_n : D_n >),$$

либо в сокращенных вариантах: $R(A_1, A_2, ..., A_n)$ или R.

Число атрибутов в отношении называется *степенью* (либо *арностью*) отношения, а множество кортежей отношения — *мощностью* отношения.

Экземпляр отношения — это множество кортежей для данного отношения. Экземпляр может изменяться с течением времени. Обычная база данных в текущий момент времени работает только с одной версией отношения. Такой экземпляр отношения называется текущим.

Схема отношения представляет собой набор заголовков отношения, входящих в базу данных, т. е. перечень имен атрибутов данного отношения с указанием домена, к которому они относятся:

$$S_R = (A_1, A_2, ..., A_n),\ A_i \subseteq D_i,\ i = \overline{1, n}$$

Если атрибуты принимают значения из одного и того же домена, то они называются θ-сравнимыми, где θ — множество допустимых операций сравнений, заданных для данного домена. Например, если домен содержит числовые данные, то для него допустимы все операции сравнения: $\theta = \{=, <>, >=, <=, <, >\}$. Однако и для доменов, содержащих символьные данные, могут быть заданы не только операции сравнения по равенству и неравенству значений. Если для данного домена задано лексикографическое упорядочение, то он также имеет полное множество операций сравнения.

Схемы двух отношений называются *эквивалентными*, если они имеют одинаковую степень, и возможно такое упорядочение имен атрибутов в схемах, что на одинаковых местах будут находиться сравнимые атрибуты, т. е. атрибуты, принимающие значения из одного домена:

Пусть $S_{R_1} = (A_1, A_2, ..., A_n)$ — схема отношения R_1. $S_{R_2} = (B_{i1}, B_{i2}, ..., B_{in})$ — схема отношения R_2 после упорядочения имен атрибутов. Тогда

$$S_{R_1} \sim S_{R_2} \Leftrightarrow \begin{cases} n = m, \\ A_j, B_{ij} \subseteq D_j. \end{cases}$$

Таким образом, для эквивалентных отношений выполняются следующие условия:

- наличие одинакового количества атрибутов;

- наличие атрибутов с одинаковыми наименованиями;
- содержание данных из одних и тех же доменов для атрибутов с одинаковыми наименованиями;
- наличие в отношениях одинаковых строк с учетом того, что порядок атрибутов может различаться;
- отношения такого рода есть различные изображения одного и того же отношения.

Свойства отношений непосредственно следуют из приведенного ранее определения отношения. В этих свойствах в основном и состоят различия между отношениями реляционной модели данных и простыми таблицами.

- Уникальность имени отношения. Имя одного отношения должно отличаться от имен других отношений.
- Уникальность кортежей. В отношении нет одинаковых кортежей. Действительно, тело отношения есть множество кортежей и, как всякое множество, не может содержать неразличимые элементы. Таблицы в отличие от отношений могут содержать одинаковые строки. Каждая ячейка отношения содержит только атомарное (неделимое) значение.
- Неупорядоченность кортежей. Кортежи не упорядочены (сверху вниз), т. к. тело отношения есть множество, а множество не упорядочено (для сравнения — строки в таблицах упорядочены). Одно и то же отношение может быть изображено разными таблицами, в которых строки идут в различном порядке.
- Неупорядоченность атрибутов. Атрибуты не упорядочены (слева направо).
- Уникальность имени атрибута в пределах отношения. Каждый атрибут имеет уникальное имя в пределах отношения, значит, порядок атрибутов не имеет значения (для сравнения — столбцы в таблице упорядочены). Это свойство несколько отличает отношение от математического определения отношения. Одно и то же отношение может быть изображено разными таблицами, в которых столбцы идут в различном порядке.

- Атомарность значений атрибутов. Все значения атрибутов атомарны. Это следует из того, что лежащие в их основе атрибуты имеют атомарные значения, т. е. с каждым атрибутом связана какая-то область значений (отдельный элементарный тип), значения атрибутов берутся из одного и того же домена. Схема и кортежи отношения — множества, а не списки, поэтому порядок их представления не имеет значения. Для сравнения — в ячейки таблицы можно поместить различную информацию: массивы, структуры, другие таблицы и т. д.

Замечание

Из свойств отношения следует, что не каждая таблица может быть отношением. Для того чтобы некоторая таблица задавала отношение, необходимо, чтобы таблица имела простую структуру (содержала только строки и столбцы, причем в каждой строке должно быть одинаковое количество полей), в таблице не должно быть одинаковых строк, любой столбец таблицы должен содержать данные только одного типа, все используемые типы данных должны быть простыми.

Следует отметить, что реляционная модель представляет собой базу данных в виде множества взаимосвязанных отношений, которые называются *схемой реляционной базы данных*.

Связи, которые можно выделить между отношениями в базе данных, классифицируются по следующим типам.

- "Один-к-одному" — один экземпляр первого отношения связан только с одним экземпляром второго отношения. Как правило, существование такого типа связи свидетельствует о том, что отношения в пределах схемы базы данных заданы некорректно.
- "Один-ко-многим" — один экземпляр первого (базового) отношения связан со многими экземплярами второго (подчиненного) отношения. Данная связь является основой зависимостей между отношениями в реляционной модели данных.
- "Многие-ко-многим" — многие экземпляры первого отношения связаны со многими экземплярами второго от-

ношения. Связь такого вида практически не реализуется в реляционной среде. В силу этого она легко разрешается введением дополнительного (производного) отношения, которое является промежуточным между исходными отношениями и соединено с ними двумя связями "один-ко-многим".

Структурная часть базы данных. Виды отношений

При рассмотрении реляционной модели можно выделить следующие объекты, которые непосредственно используются любой СУБД для оптимальной организации работы с данными.

Переменная отношения P представляет собой именованный объект, значение которого может изменяться с течением времени. Значение этой переменной в любой момент времени будет значением отношения.

Если рассматривать отношения как таблицы, то переменная отношения *P* в разное время будет представлять различные таблицы, которые характеризуются разными строками, но одинаковыми столбцами.

Именованное отношение — это переменная отношения, определенная в СУБД с помощью операторов `CREATE TABLE`, `CREATE BASE RELATION`, `CREATE VIEW` и `CREATE SNAPSHOT`.

Базовым отношением называется именованное отношение, которое не является производным, т. е. существование базового отношения не зависит от существования других отношений.

Производным отношением называется отношение, определенное посредством реляционного выражения через другие именованные отношения; производное отношение зависит от существования другого либо других базовых отношений. Не путайте данное отношение с *выражаемым*!

Выражаемое отношение — это отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения. Каждое именованное отношение является выражаемым отношением, но не наоборот. Базовые отношения, представления, снимки, промежуточные и окончательные результаты отчетов представляют собой примеры выражаемых отношений. Таким образом, множество всех выражаемых отношений — это множество всех базовых отношений и всех производных отношений.

Представление — именованное производное отношение. Представления, как и базовые отношения, являются переменными отношений. В физической памяти СУБД представление не хранится, а формируется с использованием других именованных отношений.

Снимки — именованные производные отношения, аналогичные представлениям, за исключением того, что снимки реальны и представлены не только с помощью определения в терминах других именованных отношений, но и также своими отдельными данными. Создание снимка похоже на выполнение запроса к базе данных по поводу данных, за исключением того, что результат подобного запроса сохраняется в базе данных под определенным именем как отношение, которое доступно только для чтения. Периодически снимок "обновляется", т. е. его текущее значение сбрасывается, запрос выполняется снова и его результат становится новым значением снимка.

Результатом запроса называется неименованное производное отношение, служащее результатом некоторого определенного вопроса к базе по поводу данных. База данных не обеспечивает постоянного существования для результатов запросов. Для сохранения результатов запроса его можно присвоить некоторому именованному отношению.

Промежуточным результатом называется именованное производное отношение, являющееся результатом некоторого реляционного выражения, вложенного в другое, большее выражение. База данных не поддерживает постоянного сущест-

вования для промежуточных результатов, как и для окончательных результатов.

Хранимым называется отношение, которое поддерживается в физической памяти "непосредственным" образом.

Следует отметить, что хранимое отношение не всегда совпадает с базовым отношением. Набор хранимых отношений должен быть таким, чтобы все базовые отношения, а значит, и все выражаемые отношения можно было произвести из него; но при этом требуется, чтобы все хранимые отношения были базовыми, и не требуется, чтобы все базовые отношения были хранимыми.

Итак, именованное отношение является переменной, т. е. заголовок отношения фиксирован, но значение тела изменяется со временем. Именованные отношения — это базовые отношения, представления и снимки. Неименованное отношение — результат вычисления реляционного выражения, которое обычно существует недолго и его тело не изменяется в процессе существования. Примерами неименованных отношений могут быть промежуточные и окончательные результаты запросов.

Для каждого отношения существует связанная с ним интерпретация (или *предикат*), составляющая *критерий возможности обновления* для этого отношения. В любой момент времени отношение содержит лишь те кортежи, при которых предикат является истинным.

Реляционная целостность данных

Реляционная целостность данных рассматривается в двух аспектах — ключи отношения и реляционные ограничения целостности.

Первичным ключом (ключом отношения, ключевым атрибутом) называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Ключ может быть составным (сложным), т. е. состоять из нескольких атрибутов.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Таким образом, ее существование гарантирует то, что отношение — это множество, которое не содержит одинаковых кортежей. Во многих СУБД можно создавать отношения, не определяя ключи.

Иногда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Такие комбинации атрибутов — это возможные ключи отношения и любой из них можно выбрать в качестве первичного.

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

Обычно ключи используют для:

- исключения дублирования значений в ключевых атрибутах (другие атрибуты в расчет не принимаются);
- упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним атрибутам возрастание, по другим — убывание);
- ускорения работы с кортежами отношения;
- организации связывания таблиц.

Если в отношении R_1 имеется *не ключевой* атрибут A, значения которого являются значениями *ключевого* атрибута B другого отношения R_2, то атрибут A отношения R_1 является *внешним ключом*.

С помощью внешних ключей устанавливаются связи между отношениями. Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных — ограничение ссылочной целостности, т. е. каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

Так как каждый атрибут связан с некоторым доменом, для множества допустимых значений каждого атрибута отноше-

ния определяют так называемые ограничения домена. Кроме этого, для БД задаются два правила целостности, которые называются *реляционными ограничениями целостности* — ограничениями для всех допустимых состояний БД:

- правила целостности сущностей (отношений);
- правила ссылочной целостности.

Определитель NULL указывает на то, что значение атрибута в данный момент неизвестно или неприемлемо для данного кортежа. NULL не входит в область определения некоторого кортежа либо никакое значение еще не задано. Ключевое слово NULL представляет собой способ обработки неполных или необычных данных. NULL ни в коем случае нельзя отождествлять как нулевое численное значение или заполненную пробелами текстовую строку (нули и пробелы — это некоторые значения; NULL — это отсутствие значения).

Целостность отношений — в базовом (основном) отношении ни один атрибут первичного ключа не может содержать отсутствующих значений, т. е. NULL-значений.

Ссылочная целостность — значение внешнего ключа отношения должно либо соответствовать значению первичного ключа базового отношения, либо задаваться определителем NULL.

Корпоративные ограничения целостности — дополнительные правила поддержки целостности данных, определяемые пользователями или администраторами БД.

Индексирование

Индекс представляет собой средство ускорения поиска записей в таблице, а также других операций, использующих поиск: извлечение, модификацию, сортировку и т. д. Таблицу, для которой используют индекс, называют *индексированной*.

Индекс содержит отсортированную по колонке или нескольким колонкам информацию и указывает на строки, в которых хранится конкретное значение колонки. Индекс выполняет

роль оглавления таблиц, просмотр которого предшествует обращению к записям таблицы. В некоторых системах индексы хранятся в индексных файлах отдельно от табличных.

Решение проблемы организации физического доступа к информации зависит в основном от следующих факторов:

- вида содержимого в поле ключа записей индексного файла;
- типа используемых ссылок (указателей) на запись основной таблицы;
- метода поиска нужных записей.

Индексный файл — это хранимый файл особого типа, в котором каждая запись состоит из двух значений: данное и RID-указатель. На практике чаще всего используются два метода поиска: последовательный и бинарный (основан на делении интервала поиска пополам).

Поиск необходимых записей при индексации может происходить по одноуровневой либо двухуровневой схеме индексации. При одноуровневой схеме в индексном файле хранятся короткие записи, имеющие два поля: поле содержимого старшего ключа (хеш-кода ключа) адресуемого блока и поле адреса начала этого блока (рис. 2.2.).

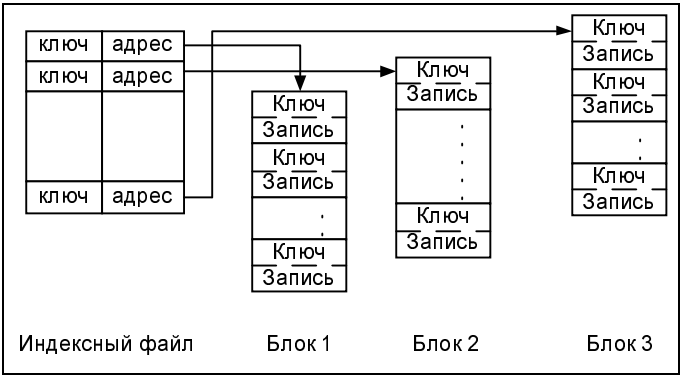


Рис. 2.2. Одноуровневая схема индексации

В каждом блоке записи располагаются в порядке возрастания значения ключа (или *свертки*). Старшим ключом каждого блока является ключ его последней записи.

Если в индексном файле хранятся хеш-коды ключевых полей индексированной таблицы, то алгоритм поиска нужной записи включает три этапа.

1. Образование свертки значения ключевого поля искомой записи.
2. Поиск в индексном файле записи о блоке, значение первого поля которого больше полученной свертки (это гарантирует нахождение искомой свертки в данном блоке).
3. Последовательный просмотр записей блока до совпадения сверток искомой записи и записи блока файла. В случае *коллизий* (нескольким разным ключам может соответствовать одно значение хеш-функции, т. е. один адрес) сверток ищется запись, значение ключа которой совпадает со значением ключа искомой записи.

Недостаток одноуровневой схемы — ключи (свертки) записей хранятся вместе с записями, что приводит к увеличению времени поиска записей из-за большой длины просмотра (значения данных в записях приходится пропускать).

В двухуровневой схеме ключи (свертки) записей отделены от содержимого записей (рис. 2.3).

В данном случае индекс основной таблицы распределен по совокупности файлов: одному файлу главного индекса и множеству файлов с блоками ключей.

На практике при создании индекса для некоторой таблицы базы данных указывают поле таблицы, которое требует индексации. Ключевые поля таблицы во многих СУБД индексируются автоматически. Индексные файлы, создаваемые по ключевым полям таблицы, называются *файлами первичных индексов*.

Индексы, создаваемые не для ключевых полей, называются вторичными (пользовательскими) индексами. Введение этих

индексов не изменяет физического расположения записей таблицы, но влияет на последовательность просмотра записей. Индексные файлы, создаваемые для поддержания вторичных индексов таблицы, называются *файлами вторичных индексов.*

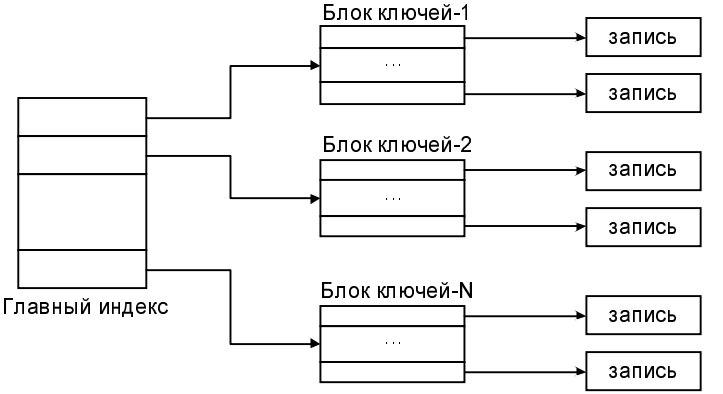


Рис. 2.3. Двухуровневая схема индексации

Некоторые СУБД поддерживают кластеризованные и кластеризованные хешированные индексы.

Кластеризация — помещение в один блок записей таблиц, которые с большой вероятностью будут часто подвергаться соединению.

Кластеризованный индекс — специальная техника индексирования, при которой данные в таблице физически располагаются в индексированном порядке. Использование кластеризованного индекса значительно ускоряет выполнение запросов по индексированной колонке. Для каждой таблицы может существовать только один кластеризованный индекс. При создании кластеризованного индекса не по первичному ключу автоматически снимается кластеризация по первичному ключу.

Хеширование — альтернативный способ хранения данных в заранее заданном порядке с целью ускорения поиска (прямой доступ). Хеширование избавляет от необходимости поддерживать и просматривать индексы. *Кластеризованный хешированный индекс* значительно ускоряет операции поиска и сортировки, но добавление и удаление строк замедляется из-за необходимости реорганизации данных для соответствия индексу. Хеширование применяется в том случае, когда необходим прямой доступ (без индексов), например при бронировании авиабилетов, мест в гостиницах, прокате машин, а также электронных денежных переводах. Однако недостатком хеширования являются необходимость нахождения соответствующей хеш-функции, необходимость выполнения операции свертки (требует определенного времени), возможные коллизии (свертка различных значений может дать одинаковый хеш-код) и промежутки между записями неопределенной протяженности. При хешировании RID-указатель вычисляется с помощью некоторой хеш-функции и называется хеш-кодом. В поле ключа индексного файла можно хранить значения ключевых полей индексируемой таблицы либо свертку ключа (хеш-код). Длина хеш-кода всегда постоянна и имеет достаточно малую величину (например, 4 байта), а это существенно снижает время поисковых операций.

Общим недостатком индексных схем является необходимость хранения индексов, к которым требуется обращаться для обнаружения записей.

Задания

1. Сформулировать основные понятия реляционной модели данных и дать соответствующую интерпретацию в табличных терминах.
2. Какие аспекты данных рассматривает реляционная модель?
3. В чем отличие типа данных от домена?
4. Что такое степень и мощность отношения?

5. В каком случае схемы двух отношений будут эквивалентными?
6. Чем отношение в реляционной модели отличается от простой таблицы? Перечислить главные отличия и привести соответствующие примеры.
7. Что такое схема базы данных? Какова может быть математическая запись схемы базы данных?
8. Какие типы связей можно выделить между отношениями в реляционной модели? Привести примеры из предметной области, отражающие возможное существование таких связей.
9. Перечислить виды отношений реляционной модели и сделать их сравнительный анализ.
10. Какие аспекты включает реляционная целостность данных? Привести соответствующие примеры.
11. Что такое индекс? В чем состоят преимущества и недостатки использования индексов?
12. Определить первичные и внешние ключи для следующих отношений, находящихся в пределах одной схемы базы данных:
    - `(Код спортивного мероприятия, Название мероприятия, Описание)`;
    - `(Код судьи, ФИО судьи, Категория, Служебный адрес, Служебный телефон)`;
    - `(Код спортивного мероприятия, Код судьи, Дата проведения)`.
13. Определить первичные и внешние ключи для следующих отношений, находящихся в пределах одной схемы базы данных:
    - `(Код товара, Название, Описание)`;
    - `(Код товара, Код продукта, Количество)`;
    - `(Код продукта, Название продукта, Цена продукта)`;

- (Код накладной, Дата выписки, Дата оплаты);
- (Код накладной, Код товара, Количество товара).

14. Определить первичные и внешние ключи для следующих отношений, находящихся в пределах одной схемы базы данных:

- (Личный номер аспиранта, Фамилия, Имя, Отчество, Пол, Дата рождения, Дата поступления, Код специальности);
- (Код специальности, Специальность, Описание);
- (Личный номер аспиранта, Тема, Научный руководитель, Дата утверждения);
- (Код дисциплины, Название дисциплины, ФИО преподавателя);
- (Личный номер аспиранта, Код дисциплины, Оценка, Дата сдачи).

Глава 3

Реляционная алгебра

Обработку данных реляционной модели можно реализовать двумя различными, но эквивалентными способами: реляционной алгеброй и реляционным исчислением. В данной главе рассмотрены начала реляционной алгебры.

Основные определения, относящиеся к реляционной алгебре

Реляционная алгебра представляет собой основу доступа к реляционным данным. Главная цель алгебры — обеспечить запись выражений, которые могут использоваться для следующих целей:

- определение данных для их выбора из базы как результат операции выборки;
- определение данных для модификации (вставки, изменения или удаления) как результат операции обновления;
- определение данных для их визуализации через представления;
- определение данных для сохранения в виде "мгновенного снимка" отношения;
- определение данных, для которых осуществляется контроль доступа (определение правил безопасности);

- определение данных, которые входят в область для некоторых операций управления одновременным доступом (определение требований устойчивости);
- определение правил целостности, т. е. некоторых особых правил, которым должна удовлетворять база данных, наряду с общими правилами, представляющими часть реляционной модели и применяемыми к каждой базе данных.

В современных СУБД, использующих реляционную модель, непосредственно ни реляционная алгебра, ни реляционное исчисление не используются. Фактическим стандартом доступа к реляционным данным является язык SQL (Structured Query Language, структурированный язык запросов).

Реляционная алгебра, определенная Коддом, состоит из 8 операторов, разделенных на две группы:

- *традиционные операции над множествами* (объединение, пересечение, вычитание, декартово произведение);
- *специальные реляционные операции* (выборка, проекция, соединение, деление).

Кроме того, в состав алгебры включается операция присваивания, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, а также операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

Замкнутость реляционной алгебры

Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов и возвращающих отношения в качестве результата. Таким образом, реляционный оператор f выглядит как функция с отношениями в качестве аргументов:

$$R = f(R_1, R_2, ..., R_n).$$

В качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу:

$$R = f\left(f_1\left(R_{11}, R_{12}, ..., R_{1n}\right),\ f_2\left(R_{21}, R_{22}, ..., R_{2m}\right),\ ...\right).$$

В силу этого реляционная алгебра является замкнутой. Таким образом, в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

В пределах базы данных каждое отношение обязано иметь уникальное имя. Имя отношения, полученного в результате выполнения реляционной операции, определяется в левой части равенства. Однако можно не требовать наличия имен от отношений, полученных в результате реляционных выражений, если эти отношения подставляются в качестве аргументов в другие реляционные выражения. Такие отношения называются *неименованными*. Неименованные отношения реально не существуют в базе данных, а лишь представляются в момент вычисления значения реляционного оператора. Не все они являются независимыми, т. е. некоторые из этих операторов могут быть выражены через другие реляционные операторы.

Отношения, совместимые по типу

Некоторые реляционные операторы (например, объединение) требуют, чтобы отношения имели одинаковые заголовки. Как указывалось в *главе 2*, отношение состоит из заголовка и тела. Операция объединения двух отношений есть объединение двух множеств кортежей, взятых из тел соответствующих отношений. Однако будет ли результат считаться отношением? Во-первых, если исходные отношения имеют разное количество атрибутов, то, очевидно, что множество, являющееся объединением таких разнотипных кортежей, нельзя представить в виде отношения. Во-вторых, пусть даже отношения имеют одинаковое количество атрибутов, но атрибуты имеют различные наименования. Тогда возникает вопрос, как тогда определить заголовок отношения, полученного в результате

объединения множеств кортежей? В-третьих, пусть отношения имеют одинаковое количество атрибутов, атрибуты имеют одинаковые наименования, но определены на различных доменах. Тогда опять же объединение кортежей не будет образовывать отношение.

Определение. Отношения называются *совместимыми по типу*, если они имеют идентичные заголовки, а именно:

- отношения имеют одно и то же множество имен атрибутов, т. е. для любого атрибута в одном отношении найдется атрибут с таким же наименованием в другом отношении;
- атрибуты с одинаковыми именами определены на одних и тех же доменах.

Некоторые отношения не являются совместимыми по типу, но становятся таковыми после некоторого переименования атрибутов. Для того чтобы такие отношения можно было использовать в реляционных операторах, вводится вспомогательный *оператор переименования атрибутов*.

Оператор переименования атрибутов

Оператор переименования атрибутов имеет следующий синтаксис:

R `rename` A_1 `,` A_2 `, ... as new` A_1 `new` A_2 `, ...;`

где R — отношение, A_1 , A_2 , ... — исходные имена атрибутов, `new` A_1 , `new` A_2 , ... — новые имена атрибутов. В результате применения оператора переименования атрибутов получаем новое отношение с измененными именами атрибутов.

Пример. Оператор `rename` возвращает неименованное отношение, в котором атрибут `Student` переименован в `Head` (Староста):

R `rename Student as Head;`

Традиционные операции над множествами (теоретико-множественные операторы)

К традиционным операциям реляционной алгебры относят классические операции над множествами, т. е. объединение, пересечение, вычитание и декартово произведение. Рассмотрим результаты действий данных операций над множествами отношений в реляционной модели данных.

Объединение

Определение. *Объединением* двух совместимых по типу отношений R_1 и R_2 называется отношение с тем же заголовком, что и у отношений R_1 и R_2, и телом, состоящим из кортежей, принадлежащих или R_1, или R_2, или обоим отношениям. Таким образом, при выполнении операции объединения двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов. Синтаксис операции объединения:

R_1 `union` R_2

Замечание

Объединение, как и любое отношение, не может содержать одинаковых кортежей. В силу этого, если некоторый кортеж входит и в отношение R_1, и отношение R_2, то в объединение он входит только один раз.

Пример. Пусть даны два отношения R_1 и R_2 с информацией о начислении стипендии студентам (табл. 3.1 и 3.2).

Таблица 3.1. *Отношение* R_1 *(Начисление стипендии)*

Личный номер	Фамилия	Размер стипендии
11	Котова	3000
22	Серов	2500
33	Леонидов	3500

Таблица 3.2. *Отношение* R_2 *(Начисление стипендии)*

Личный номер	Фамилия	Размер стипендии
11	Котова	3000
22	Даниленко	2500
55	Леонидов	3000

Объединение отношений R_1 и R_2 будет иметь вид, представленный в табл. 3.3.

Таблица 3.3. *Отношение* R_1 `union` R_2

Личный номер	Фамилия	Размер стипендии
11	Котова	3000
22	Серов	2500
33	Леонидов	3500
22	Даниленко	2500
55	Леонидов	3000

Замечание

Реляционные операторы не передают результирующему отношению никаких данных о потенциальных ключах в силу того, что потенциальный ключ является семантическим понятием, отражаю-

щим различия объектов предметной области. Наличие потенциальных ключей не выводится из структуры отношения, а явно задается для каждого отношения, исходя из его смысла. Реляционные операторы являются формальными операциями над отношениями и выполняются одинаково, независимо от смысла данных, содержащихся в отношениях.

В силу сказанного операция объединения (а также и другие операции реляционной алгебры) не наследует потенциальные ключи отношений, входящих в объединение. Поэтому в объединении отношений R_1 и R_2 атрибут `Личный номер` может содержать дубликаты значений. Следует отметить, что объединение отношений R_1 и R_2 как и любое отношение, имеет потенциальный ключ, например, состоящий из атрибутов `(Личный номер, Фамилия)`.

Пересечение

Определение. *Пересечением* двух совместимых по типу отношений R_1 и R_2 называется отношение с тем же заголовком, что и у отношений R_1 и R_2, и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям R_1 и R_2. Таким образом, операция пересечения двух отношений дает отношение, включающее все кортежи, входящие в оба отношения-операнда. Синтаксис операции пересечения:

R_1 `intersect` R_2

Пример. Для отношений R_1 и R_2 (см. табл. 3.1 и 3.2) пересечение имеет вид, представленный в табл. 3.4.

Таблица 3.4. *Отношение* R_1 `intersect` R_2

Личный номер	Фамилия	Размер стипендии
11	Котова	3000

Вычитание

Определение. *Вычитанием* двух совместимых по типу отношений R_1 и R_2 называется отношение с тем же заголовком, что и у отношений R_1 и R_2, и телом, состоящим из кортежей, принадлежащих отношению R_1 и не принадлежащих отношению R_2. Таким образом, отношение, которое является разностью двух отношений, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, являющееся вторым операндом. Синтаксис операции вычитания:

R_1 `minus` R_2

Пример. Для отношений R_1 и R_2 (см. табл. 3.1 и 3.2) вычитание имеет вид, представленный в табл. 3.5.

Таблица 3.5. *Отношение* R_1 `minus` R_2

Личный номер	Фамилия	Размер стипендии
22	Серов	**2500**
33	**Леонидов**	**3500**

Декартово произведение

Определение. *Декартовым произведением* двух отношений $R_1(R_{11}, R_{12}, ..., R_{1n})$ и $R_2(R_{21}, R_{22}, ..., R_{2m})$ называется отношение, заголовок которого является *сцеплением заголовков* отношений R_1 и R_2:

$$(R_{11}, R_{12}, ..., R_{1n}, R_{21}, R_{22}, ..., R_{2m}),$$

а тело состоит из кортежей, являющихся *сцеплением кортежей* отношений R_1 и R_2:

$$(r_{11}, r_{12}, ..., r_{1n}, r_{21}, r_{22}, ..., r_{2m})$$

таких, что:

$$(r_{11}, r_{12}, ..., r_{1n},) \in R_1, \ (r_{21}, r_{22}, ..., r_{2m}) \in R_2.$$

При выполнении прямого произведения двух отношений получается отношение, кортежи которого являются конкатенацией (сцеплением) кортежей первого и второго операндов. Синтаксис операции декартового произведения:

R_1 `times` R_2

Замечание

Мощность произведения R_1 `times` R_2 равна произведению мощностей отношений R_1 и R_2, т. к. каждый кортеж отношения R_1 соединяется с каждым кортежем отношения R_2.

Если в отношениях R_1 и R_2 имеются атрибуты с одинаковыми наименованиями, то перед выполнением операции декартового произведения такие атрибуты необходимо переименовать. При перемножении отношений совместимость по типу не требуется.

Пример. Пусть даны два отношения R_1 и R_2 с информацией о студентах и дисциплинах (табл. 3.6 и 3.7).

Таблица 3.6. *Отношение R_1 (Студенты)*

Личный номер	Фамилия студента
11	Котова
22	Серов
3	Леонидов

Таблица 3.7. *Отношение* R_2 *(Название дисциплины)*

Код дисциплины	Название дисциплины
1	Высшая математика
2	Иностранный язык
3	Философия

Декартово произведение отношений R_1 и R_2 будет иметь вид в соответствии с табл. 3.8.

Таблица 3.8. *Отношение* R_1 `times` R_2

Личный номер	Фамилия студента	Код дисциплины	Название дисциплины
11	Котова	1	Высшая математика
11	Котова	2	Иностранный язык
11	Котова	3	Философия
22	Серов	1	Высшая математика
22	Серов	2	Иностранный язык
22	Серов	3	Философия
33	Леонидов	1	Высшая математика
33	Леонидов	2	Иностранный язык
33	Леонидов	3	Философия

Замечание

Операция декартова произведения непосредственно для реальных запросов не используется, однако она важна для выполнения специальных реляционных операций.

Специальные реляционные операторы

Специальные реляционные операции (выборка, проекция, соединение, деление) предназначены непосредственно для работы с отношениями реляционной модели, и их действие не распространяется на произвольные множества (как в случае с традиционными операциями).

Выборка (ограничение, селекция)

Определение. *Выборкой (ограничением, селекцией)* на отношении R с условием c называется отношение с тем же заголовком, что и у отношения R, и телом, состоящим из кортежей, значения атрибутов которых при подстановке в условие c дают значение ИСТИНА. Условие c представляет собой логическое выражение, в которое могут входить атрибуты отношения R и (или) скалярные выражения. Таким образом, результатом ограничения отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.

Обычно условие c имеет вид $R_{11}\Theta R_{12}$, где $\Theta \in \{=, \neq, <, >, \leq, \geq\}$, а R_{11} и R_{12} — атрибуты отношения R или скалярные значения. Такие выборки называются Θ-*выборки (тэта-выборки)* или Θ-*ограничения*, Θ-*селекции*.

Синтаксис операции выборки:

R `where` c, или R `where` $R_{11}\Theta R_{12}$.

Пример. Пусть дано отношение R с информацией (табл. 3.9).

Таблица 3.9. *Отношение R (Список студентов и их стипендий)*

Личный номер	Фамилия студента	Размер стипендии
1	Котова	3000

Таблица 3.9 *(окончание)*

Личный номер	Фамилия студента	Размер стипендии
2	Серов	2500
3	Леонидов	3500

Результат выборки R `where Размер стипендии > 2500` представлен в табл. 3.10.

Таблица 3.10. *Отношение* R `where Размер стипендии > 2500`

Личный номер	Фамилия студента	Размер стипендии
1	Котова	3000
3	Леонидов	3500

Таким образом, операция выборки позволяет получить кортеж отношения, удовлетворяющий определенным условиям, в результате чего получается *"горизонтальный срез"* отношения по некоторому условию.

Проекция

Определение. *Проекцией* отношения R по атрибутам $R_{11}, R_{12}, ..., R_{1n}$, где каждый из атрибутов принадлежит отношению R, называется отношение с заголовком $(R_{11}, R_{12}, ..., R_{1n})$ и телом, содержащим множество кортежей вида $(r_{11}, r_{12}, ..., r_{1n})$ таких, для которых в отношении R найдутся кортежи со значением атрибута R_1, равным r_1, значением атрибута R_2, равным r_2, ..., значением атрибута R_n, равным r_n. Таким образом, при выполнении проекции отношения на заданный набор его атрибутов получается отношение, кортежи которого производятся путем взятия соответствующих значе-

ний из кортежей отношения-операнда. Синтаксис операции проекции:

$$R[R_1, R_2, ..., R_n].$$

Замечание

Операция проекции получает *"вертикальный срез"* отношения, в котором удалены все возникшие дубликаты кортежей.

Пример. Пусть дано отношение R с информацией (табл. 3.11).

Таблица 3.11. *Студенты факультетов*

Личный номер	Фамилия студента	Факультет
11	Котова	Исторический
22	Серов	Математический
33	Леонидов	Исторический
55	Серов	Физический

Результаты операции проекции R `[Факультет]` представлены в табл. 3.12.

Таблица 3.12. *Отношение* R `[Факультет]`

Факультет
Исторический
Математический
Физический

Соединение

Операция соединения отношений, наряду с операциями выборки и проекции, является одной из наиболее важных реля-

ционных операций. При соединении двух отношений по некоторому условию образуется результирующее отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию. Обычно рассматривается несколько разновидностей операции соединения:

- общая операция соединения;
- Θ-соединение (тэта-соединение);
- экви-соединение;
- естественное соединение.

Наиболее важным из данных частных случаев является операция естественного соединения, т. к. остальные разновидности соединения являются частными случаями общей операции соединения.

Общая операция соединения

Определение. *Соединением* отношений R_1 и R_2 по условию c называется отношение:

(R_1 times R_2) where c,

где c представляет собой логическое выражение, в которое могут входить атрибуты отношений R_1 и R_2 и/или скалярные выражения.

Таким образом, операция соединения есть результат последовательного применения операций декартова произведения и выборки. Если в отношениях R_1 и R_2 имеются атрибуты с одинаковыми наименованиями, то перед выполнением соединения такие атрибуты необходимо переименовать.

Тэта-соединение (Θ-join)

Определение. Пусть отношение R_1 содержит атрибут R_{11}, отношение R_2 содержит атрибут R_{21}, а Θ — один из операто-

ров сравнения $\Theta \in \{=, \neq, <, >, \leq, \geq\}$. Тогда Θ-*соединением* отношения R_1 по атрибуту R_{11} с отношением R_2 по атрибуту R_{21} называют отношение

(R_1 `times` R_2) `where` $R_{11}\,\Theta\,R_{21}$.

Это частный случай операции общего соединения. Нередко для операции Θ-соединения применяют более короткий синтаксис:

R_1 [$R_{11}\,\Theta\,R_{21}$] R_2.

Пример. Рассмотрим деятельность деканата, которому необходимо хранить данные о студентах и сдаваемых ими дисциплинах. Каждый студент обучается на определенном курсе (табл. 3.13). Каждая дисциплина читается в определенном семестре (табл. 3.14). Таким образом, в деканате должны храниться сведения о сдаче студентами соответствующих экзаменов, семестр которых соответствует определенному курсу (например, при сдаче летней экзаменационной сессии закономерность будет такова: `семестр = 2*курс`; при сдаче зимней `семестр = 2*курс - 1`).

Таблица 3.13. *Отношение* R_1 *(Студенты)*

Личный номер	Фамилия студента	Курс
11	Котова	4
22	Серов	1
33	Леонидов	3

Таблица 3.14. *Отношение* R_2 *(Дисциплины)*

Код дисциплины	Название дисциплины	Семестр
1	Высшая математика	2
2	Иностранный язык	3

Таблица 3.14 *(окончание)*

Код дисциплины	Название дисциплины	Семестр
3	Философия	1
4	Психология	9
5	Спецкурс	10

Результаты вопроса (запроса) "Какие экзамены сданы всеми студентами, которые закончили учебный год.", т. е. R_1 `[семестр ≤ 2 * курс]` R_2 приведены в табл. 3.15.

Таблица 3.15. *Отношение "Какие дисциплины сдали студенты, которые закончили учебный год"*

Личный номер	Фамилия студента	Курс	Код дисциплины	Название дисциплины	Семестр
11	Котова	4	1	Высшая математика	2
11	Котова	4	2	Иностранный язык	3
11	Котова	4	3	Философия	1
22	Серов	1	1	Высшая математика	1
22	Серов	1	3	Философия	2
33	Леонидов	3	1	Высшая математика	2
33	Леонидов	3	2	Иностранный язык	3
33	Леонидов	3	3	Философия	1

Экви-соединение

Наиболее важный частный случай Θ-соединения — это когда Θ есть просто равенство. Синтаксис *экви-соединения*:

$$R_1 \ [\ R_{11} = R_{21}\ ] \ R_2$$

Пример. Пусть имеются отношения R_1, R_2 и R_3, хранящие информацию о студентах, дисциплинах и оценках соответственно (табл. 3.16—3.18).

Таблица 3.16. *Отношение R_1 (Студенты)*

Личный номер	Фамилия студента
11	Котова
22	Серов
33	Леонидов

Таблица 3.17. *Отношение R_2 (Дисциплины)*

Код дисциплины	Название дисциплины
1	Высшая математика
2	Иностранный язык
3	Философия

Таблица 3.18. *Отношение R_3 (Оценки)*

Личный номер	Код дисциплины	Оценка
11	1	5
11	2	4
11	3	5
22	1	3

Таблица 3.18 *(окончание)*

Личный номер	Код дисциплины	Оценка
22	3	4
33	2	5

Если необходимо получить из данных отношений информацию, касающуюся всех оценок, полученных студентами, то нужно использовать экви-соединение:

R_1 [Личный_номер = Личный_номер] R_3.

Так как в отношениях имеются одинаковые атрибуты, то сначала необходимо переименовать атрибуты, а потом выполнить экви-соединение:

(R_1 rename Личный_номер as Личный_номер_1)[Личный_номер_1 = Личный_номер_2]

(R_3 rename Личный_номер as Личный_номер_2)

Но, как правило, такой сложной формой записи не пользуются. Итак, результатом такого экви-соединения является отношение, представленное в табл. 3.19.

Таблица 3.19. *Отношение "Какие оценки получены студентами"*

Личный номер (Личный номер 1)	Фамилия студента	Личный номер (Личный номер 2)	Код дисциплины	Оценка
11	Котова	11	1	5
11	Котова	11	2	4
11	Котова	11	3	5
22	Серов	22	1	3
22	Серов	22	3	4
33	Леонидов	33	2	5

Замечание

Недостатком экви-соединения является то, что при соединении отношений с одинаковыми атрибутами в результирующем отношении появляются два атрибута с одинаковыми значениями. Избавиться от этого недостатка можно, взяв проекцию по всем атрибутам, кроме одного из дублирующих. Именно так действует естественное соединение.

Естественное соединение (natural-join)

Определение. Пусть даны отношения $R_1(R_{11}, R_{12}, ..., R_{1n}, Z_1, Z_2, ..., Z_p)$ и $R_2(Z_1, Z_2, ..., Z_p, R_{21}, R_{22}, ..., R_{2m})$, имеющие одинаковые атрибуты $Z_1, Z_2, ..., Z_p$ (т. е. атрибуты с одинаковыми именами и определенные на одинаковых доменах). Тогда *естественным соединением* отношений R_1 и R_2 называется отношение с заголовком $(R_{11}, R_{12}, ..., R_{1n}, Z_1, Z_2, ..., Z_p, R_{21}, R_{22}, ..., R_{2m})$ и телом, содержащим множество кортежей $(r_{11}, r_{12}, ..., r_{1n}, z_1, z_2, ..., z_p, r_{21}, r_{22}, ..., r_{2m})$ таких, что $(r_{11}, r_{12}, ..., r_{1n}, z_1, z_2, ..., z_p) \in R_1$ и $(z_1, z_2, ..., z_p, r_{21}, r_{22}, ..., r_{2m}) \in R_2$.

Для обозначения естественного соединения используется следующий синтаксис:

R_1 `join` R_2

Замечание

В синтаксисе естественного соединения не указывается, по каким атрибутам производится соединение, т. к. естественное соединение производится *по всем* одинаковым атрибутам.

Естественное соединение эквивалентно следующей последовательности реляционных операций:

- переименовать одинаковые атрибуты в отношениях;
- выполнить декартово произведение отношений;
- выполнить выборку по совпадающим значениям атрибутов, имевших одинаковые имена;

- выполнить проекцию, удалив повторяющиеся атрибуты;
- переименовать атрибуты, вернув им первоначальные имена.

Можно выполнять последовательное естественное соединение нескольких отношений. Легко заметить, что естественное соединение (как, в принципе, и соединение общего вида) обладает свойством *ассоциативности*:

$$(R_1 \text{ join } R_2) \text{ join } R_3 = R_1 \text{ join } (R_2 \text{ join } R_3),$$

в силу чего такие соединения можно записывать в виде:

$$R_1 \text{ join } R_2 \text{ join } R_3.$$

Пример. Для предыдущего примера ответ на вопрос "Какими студентами получены какие оценки?" можно записать в виде естественного соединения трех отношений R_1 join R_2 join R_3 (см. результат в виде табл. 3.20).

Таблица 3.20. *Отношение* R_1 *join* R_3 *join* R_2

Личный номер	Фамилия студента	Код дисциплины	Название дисциплины	Оценка
11	Котова	1	Высшая математика	5
11	Котова	2	Иностранный язык	4
11	Котова	3	Философия	5
22	Серов	1	Высшая математика	3
22	Серов	3	Философия	4
33	Леонидов	2	Иностранный язык	5

Деление

У операции реляционного деления два операнда — бинарное и унарное отношения. Результирующее отношение состоит из одноатрибутных кортежей, включающих значения первого

атрибута кортежей первого операнда, таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) совпадает со множеством значений второго операнда.

Определение. Пусть даны отношения $R_1(R_{11}, R_{12}, ..., R_{1n}, R_{21}, R_{22}, ..., R_{2m})$ и $R_2(R_{21}, R_{22}, ..., R_{2m})$, причем атрибуты $R_{21}, R_{22}, ..., R_{2m}$ — общие для двух отношений. *Делением отношений* R_1 на R_2 называется отношение с заголовком $R_{11}, R_{12}, ..., R_{1n}$ и телом, содержащим множество кортежей $(r_{11}, r_{12}, ..., r_{1n})$, таких, что для *всех* кортежей $(r_{21}, r_{22}, ..., r_{2m}) \in R_2$ в отношении R_1 найдется кортеж $(r_{11}, r_{12}, ..., r_{1n}, r_{21}, r_{22}, ..., r_{2m})$.

Отношение R_1 является *делимым*, отношение R_2 — *делителем*. Деление отношений аналогично делению чисел с остатком. Синтаксис операции деления:

R_1 `divideby` R_2

Замечание

Типичные запросы, реализуемые с помощью операции деления, в своей формулировке обычно имеют слово "все", например "Кто из студентов сдал все экзамены?"

Пример. Рассмотрим запрос, связанный с выяснением информации, касающейся "Кто из студентов сдал все экзамены?"

В качестве делимого выбирается проекция $X = R_3$ `[Личный номер, Код дисциплины]`, которая содержит личный номер студента и код сданной им дисциплины (табл. 3.21).

Таблица 3.21. *Проекция* $X = R_3$ `[Личный номер, Код дисциплины]`

Личный номер	Код дисциплины
11	1
11	2

Таблица 3.21 *(окончание)*

Личный номер	Код дисциплины
11	3
22	1
22	3
33	2

В качестве делителя берется проекция `Y` $= R_2$ `[Код дисциплины]`, содержащая список номеров *всех* деталей (табл. 3.22):

Таблица 3.22. *Проекция* `Y` $= R_2$ `[Код дисциплины]`

Код дисциплины
1
2
3

Деление `X divideby Y` дает список номеров студентов, которые сдали все экзамены (табл. 3.23).

Таблица 3.23. *Отношение* `X divideby Y`

Личный номер
1

Итак, как оказалось, что только один студент с личным номером 1 сдал все экзамены.

Примеры использования реляционных операторов

Пример. Получить фамилии студентов, сдавших дисциплину с кодом 3.

Решение.

((R3 join R1) where Код_дисциплины = 3)[Фамилия_студента]

Пример. Получить фамилии студентов, сдавших философию.

Решение.

(((R2 where Название_дисциплины = Философия) join R3) join R1)[Фамилия_студента]

Пример. Получить фамилии студентов, сдавших все дисциплины.

Решение.

((R3[Личный_номер, Код_дисциплины] divideby R2[Код_дисциплины]) join R1)[Фамилия_студента]

Пример. Получить фамилии студентов, не сдавших дисциплину с кодом 3.

Решение.

((R1[Личный_номер] minus ((R1 join R3) where Код_дисциплины = 3)[Личный_номер]) join R1)[Фамилия_студента]

Решение данной задачи можно получить также и следующим образом:

- $S_1 = R_1$[Личный_номер] — получение списка личных номеров всех студентов;
- $S_2 = R_1$ join R_3 — соединение данных о студентах и оценках;

- $S_3 = S_2$ `where Код_дисциплины = 3` — в данных о студентах и оценках остается только информация о сдаче дисциплины с кодом 3;
- $S_4 = S_3$ `[Личный_номер]` — получение списка личных номеров студентов, сдавших дисциплину с кодом 3;
- $S_5 = S_1$ `minus` S_4 — получение списка личных номеров студентов, не сдавших дисциплину с кодом 3;
- $S_6 = S_5$ `join` R_1 — соединение списка личных номеров студентов, не сдавших дисциплину с кодом 3 с данными о студентах;
- $S_7 = S_6$ `[Фамилия_студента]` — фамилии студентов, не сдавших дисциплину с кодом 3.

Внешние соединения

При соединении двух отношений могут возникать различные ситуации, связанные с выполнением декартового произведения кортежей данных отношений: не всегда кортежу одного отношения может соответствовать совпадающее значение кортежа другого отношения.

Рассмотренное ранее естественное соединение (natural-join) является *внутренним соединением*, т. к. результирующее отношение содержит лишь значения, которые соответствуют совпадающим кортежам обоих отношений, участвующих в соединении. Часто вместо синтаксиса:

R_1 `join` R_2

можно использовать синтаксис:

R_1 `inner join` R_2

Левым (естественным) внешним соединением называется соединение, при котором кортежи отношения R_1, не имеющие совпадающих значений в общих столбцах отношения R_2,

также включаются в результирующее отношение. Для обозначения отсутствующих значений во втором отношении используется определитель NULL. Синтаксис операции левого внешнего соединения:

R_1 left join R_2

По аналогии определяется *правое (естественное) внешнее соединение*, в результате которого получается отношение, содержащее все кортежи правого отношения, синтаксис:

R_1 right join R_2

В результате выполнения *полного внешнего (естественного) соединения* получается отношение, содержащее кортежи обоих отношений, независимо от наличия совпадающих значений. Для обозначения несовпадающих значений используется определитель NULL.

Задания

1. Какие операторы составляют традиционные операции реляционной алгебры?
2. Какие операторы составляют специальные операции реляционной алгебры?
3. В чем заключается замкнутость реляционной алгебры?
4. Какие отношения называются совместимыми по типу?
5. Для чего используется оператор rename?
6. Дать характеристику традиционным операциям реляционной алгебры и привести соответствующие примеры использования операторов.
7. Дать характеристику специальным операциям реляционной алгебры и привести соответствующие примеры использования операторов.
8. Привести примеры использования различных видов внешних соединений.

9. Заданы отношения, отражающие предметную область, относящуюся к деятельности спортивного клуба (рис. 3.1).

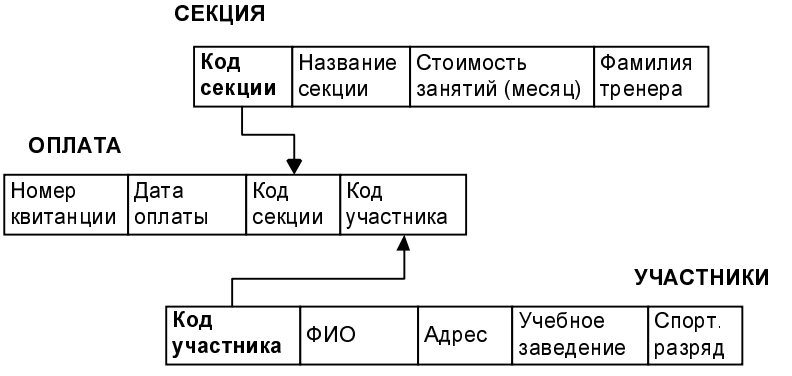


Рис. 3.1. Схема базы данных "Спортивный клуб"

С помощью языка реляционной алгебры выполнить следующие запросы:

- получить список секций и соответствующих фамилий тренера;
- получить список всех участников, которые посещают занятия в спортивном клубе с выводом следующих полей: `ФИО`, `Название секции`;
- получить список всех участников, которые посещают занятия в группах по шейпингу и плаванию;
- получить список всех участников, которые посещают занятия у определенного тренера;
- получить список всех участников, которые посещают занятия у определенного тренера и обучаются в некотором учебном заведении;
- получить список всех участников, которые произвели оплату в течение последней недели.

10. Заданы отношения, отражающие предметную область, относящуюся к деятельности цветочного магазина (рис. 3.2).

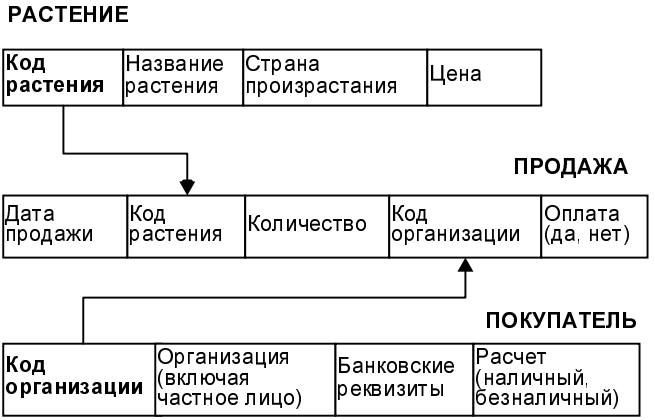


Рис. 3.2. Схема базы данных "Цветочный магазин"

С помощью языка реляционной алгебры выполнить следующие запросы:

- определить растения из любой одной страны;
- определить цену растений для данной страны произрастания;
- определить общую стоимость растений на конкретную дату продажи;
- получить список частных лиц, которые произвели покупки растений определенных наименований;
- определить организации, их банковские реквизиты, которые не произвели оплату.

Глава 4

Основы языка SQL

Стандарт ANSI для языка SQL

Для работы с базами данных, которые полностью либо частично используют реляционную модель данных, предназначен язык SQL (Structured Query Language, структурированный язык запросов).

SQL был разработан на основе реляционной алгебры и относится к непроцедурным языкам программирования — с помощью SQL можно определить то, что необходимо выполнить (а не как это делать). Суть всех операций, которые выражаются с использованием языка SQL, составляет какое-либо действие над множествами, состоящими из кортежей. Таким образом, SQL оперирует отношениями, и в результате соответствующих инструкций SQL также получается отношение.

Первый международный стандарт SQL (SQL-89 или SQL1) был разработан в 1989 году. Национальным институтом стандартизации США (American National Standards Institute, ANSI) и признан Международной организацией по стандартизации (International Organization for Standardization, ISO). В конце 1992 года появилась вторая версия SQL (SQL-92 или SQL2) и, наконец, в 1999 году принят стандарт SQL3.

Несмотря на требования спецификаций баз данных придерживаться стандартов ANSI/ISO SQL, многие разработчики

СУБД предлагают свои расширения языка SQL. Однако, несмотря на существование различных диалектов SQL, различия между конкретными расширениями языка SQL минимальны. В силу этого основные типы команд SQL практически идентичны, что, несомненно, облегчает понимание инструкций языка SQL в той либо иной СУБД конкретного разработчика.

Существуют две формы языка SQL.

Интерактивный SQL — используется для создания запросов и получения результатов в интерактивном режиме.

Встроенный SQL — включает команды SQL, которые встроены внутрь программ, написанных на другом языке программирования. Это позволяет наиболее эффективно разрабатывать приложения, которые используют данные, хранящиеся в базе.

Рассмотрим возможности интерактивного SQL, позволяющего определять данные для хранения в базе данных, манипулировать (т. е. удалять, добавлять и обновлять) ими, а также выполнять запросы к базе, т. е. получать информацию из базы данных в соответствии с некоторыми критериями отбора.

Типы команд SQL

В языке SQL можно выделить 5 типов команд, которые используются для различных целей (табл. 4.1).

Таблица 4.1. *Типы команд SQL*

Команда	Описание
DDL (Data Definition Language, язык определения данных). Позволяет создавать различные объекты базы данных и переопределять их структуру	
`CREATE TABLE`	Создание новой таблицы в базе данных
`ALTER TABLE`	Изменение структуры существующей таблицы в базе данных, включая ограничения, заданные декларативно

Таблица 4.1 *(продолжение)*

Команда	Описание
`DROP TABLE`	Удаление таблицы из базы данных
`CREATE VIEW`	Создание представления
`ALTER VIEW`	Изменение представления, созданного ранее
`DROP VIEW`	Удаление представления
`CREATE INDEX`	Создание индекса для определенной таблицы с целью обеспечения быстрого доступа по атрибутам, которые определены этим индексом
`DROP INDEX`	Удаление индекса
DML (Data Manipulation Language, язык манипулирования данными). Позволяет пользователю манипулировать данными внутри объектов реляционных баз данных	
`INSERT`	Вставка одной строки в таблицу
`UPDATE`	Обновление значения одного либо нескольких столбцов в одной или нескольких строках таблицы
`DELETE`	Удаление одной либо нескольких строк из таблицы
DQL (Data Query Language, язык запросов к данным). Позволяет выполнить выборку данных из базы в соответствии с заданными критериями	
`SELECT`	Конструирование запросов к базе данных любой сложности. Данная команда имеет много опций и необязательных параметров
DCL (Data Control Language, язык управления данными либо команды администрирования данных). Позволяет осуществлять контроль над возможностью доступа к данным внутри базы данных. Команды DCL обычно используются для создания объектов, относящихся к управлению доступом пользователем к базе данных, а также для назначения пользователям подходящих уровней привилегий (прав) доступа	
`ALTER DATABASE`	Изменение набора основных объектов и ограничений всей базы данных

Таблица 4.1 *(продолжение)*

Команда	Описание
ALTER DBAREA	Изменение созданной области хранения данных
ALTER PASSWORD	Изменение пароля всей базы данных
CREATE DATABASE	Создание новой базы данных
CREATE DBAREA	Создание области хранения базы данных
DROP DATABASE	Удаление базы данных
DROP DBAREA	Удаление области хранения базы данных
CREATE SYNONYM	Создание синонима (псевдонима) базы данных
GRANT	Предоставление прав доступа для действий над заданными объектами базы данных
REVOKE	Лишение прав доступа к объекту либо некоторым действиям над заданными объектами базы данных
Команды администрирования данных Предоставляют возможность выполнять аудит и анализ операций внутри базы данных. Эти команды могут также помочь при анализе производительности системы данных в целом. Не следует путать администрирование данных с администрированием всей БД. *Администрирование базы данных* — это осуществление общего управления базой данных, предполагающее возможность использования команд любого уровня	
START AUDIT	Начало аудита и анализа операций внутри базы данных
STOP AUDIT	Завершение аудита и анализа операций внутри базы данных
Команды управления транзакциями Позволяют выполнить обработку информации, объединенной в транзакцию	
COMMIT	Сохранение транзакции, т. е. завершение взаимосвязанной обработки информации, и тем самым изменение состояния базы данных

Таблица 4.1 *(окончание)*

Команда	Описание
ROLLBACK	Отмена транзакции, т. е. отменить изменения, которые выполнялись в ходе транзакции
SAVE POINT	Создание точки отката внутри групп транзакций, к которым отсылает команда ROLLBACK
SET TRANSACTION	Назначение имени транзакции
Процедурный SQL	
DECLARE	Определение курсора для запроса, т. е. задание имени и определение связанного с ним запроса к базе данных
OPEN	Открытие курсора, т. е. формирование виртуального набора данных, соответствующего описанию данного курсора и текущему состоянию базы данных
FETCH	Считывание очередной строки, которая задана параметром команды из виртуального набора данных, соответствующего открытому курсору
CLOSE	Закрытие курсора, т. е. прекращение доступа к виртуальному набору данных, который соответствует указанному курсору
PREPARE	Генерация плана выполнения запроса, который соответствует заданному оператору SQL
EXECUTE	Выполнение оператора SQL, который подготовлен к динамическому выполнению

Сеанс SQL

Сеанс начинается в момент подключения пользователя к базе данных:

```
CONNECT user@database
```

и введения соответствующего пароля для этого имени пользователя.

Для отключения пользователя от БД используется команда: `DISCONNECT`.

Инструкции SQL

В SQL существует около 40 инструкций, каждая из которых требует выполнить определенное действие, например, извлечь данные, создать таблицу и т. д. Все инструкции SQL имеют одинаковую структуру (рис. 4.1), заканчиваются они точкой с запятой (`;`).

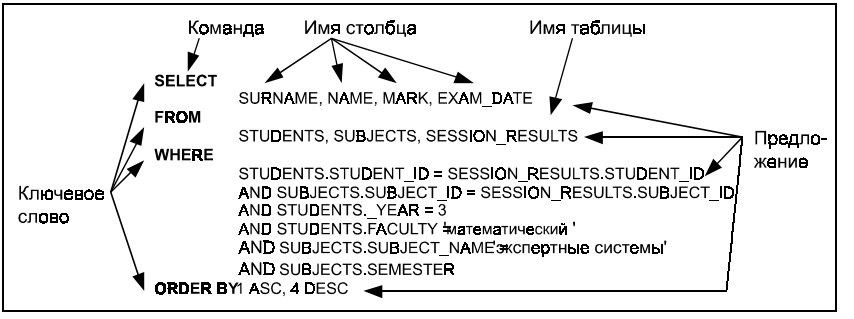


Рис. 4.1. Пример инструкции на языке SQL

Каждая инструкция SQL начинается с *команды*, т. е. *ключевого слова*, описывающего действие, выполняемое инструкцией. Командами, например, являются `CREATE`, `INSERT`, `DELETE`, `SELECT` и т. д.

После команды идет одно или несколько *предложений*, описывающих данные, с которыми работает инструкция, либо содержится уточняющая информация о действии, выполняемом инструкцией. Каждое предложение также начинается с

ключевого слова (например, `WHERE`, `FROM`, `INTO` и т. д.). Конкретная структура и содержимое предложения могут изменяться.

У каждого объекта в базе данных (схемы, таблицы, пользователи, ограничения, домены и т. д.) имеется уникальное имя. Поддержка имен в различных СУБД реализована по-разному. Рекомендуется делать имена достаточно короткими, информативными, избегать употребления в них специальных символов и пробелов, начинать с буквы.

Если в инструкции SQL указано имя таблицы, СУБД предполагает обращение к таблице текущей базы данных. Для обращения к таблицам из других баз данных (для этого необходимо иметь соответствующее разрешение) надо использовать *полное имя таблицы*, которое состоит из имени владельца (или имени схемы) и собственно имени таблицы, разделенных точкой.

Если в инструкции используется имя столбца, то СУБД сама определяет, в какой из указанных в этой же инструкции таблиц содержится данный столбец. Если необходимо включать столбцы с одинаковыми названиями из различных таблиц, то следует указывать *полные имена столбцов*, которые состоят из имени таблицы, содержащей столбец, и имени столбца (*короткого имени*), разделенных точкой. Если столбец находится в другой базе данных, следует в полном имени столбца указывать полное имя таблицы. Полное имя столбца можно использовать во всех инструкциях SQL вместо короткого имени.

Типы данных

В SQL имеются средства, позволяющие для каждого атрибута отношения указывать соответствующий тип данных. Определение типов данных в конкретных СУБД не всегда согласуется с требованиями стандарта ANSI/ISO.

Таблица 4.2. *Наиболее употребительные типы данных SQL*

Тип данных	Описание

`CHARACTER (n)`
`CHAR(n)`
где `n` — число, задающее максимальную длину соответствующего поля
	Строки символов фиксированной длины. Данный тип предполагает хранение буквенно-числовых данных. При использовании строк фиксированной длины для заполнения незанятых позиций обычно применяются пробелы. Не следует использовать этот тип для полей, в которых предполагается хранение данных переменной длины (например, фамилии), т. к. при этом нерационально используется имеющееся пространство и могут возникнуть проблемы с организацией сравнения содержащихся в соответствующих полях данных

`CHARACTER VARYING (n)`
`CHAR VARYING (n)`
`VARCHAR (n)`
`n` — максимальное число позиций, выделяемое для поля
	Строки символов переменной длины. Для данных в виде строк различной длины всегда следует использовать этот тип данных. Строки переменной длины не предполагают заполнение пробелами оставшихся позиций

`INTEGER` `INT`	Целые числа
`SMALLINT`	Малые целые числа
`BIT (n)` `n` — максимальная длина соответствующего поля	Цепочки битов постоянной длины
`BIT VARYING (n)` `n` — максимальная длина	Цепочки битов переменной длины
`NUMBER (n)` где `n` — максимальное верхнее значение	Общий числовой тип данных (для всех реализаций SQL). Числовые значения могут быть нулевыми, положительными, отрицательными, с фиксированной или плавающей точкой. Например, `NUMBER (5)`. В данном случае вводимые в поле значения ограничиваются сверху максимальным значением 99 999

Таблица 4.2 *(продолжение)*

Тип данных	Описание

`DECIMAL (p,s)`
`DEC (p,s)`
`p` — точность (общая длина числового значения).
`s` — степень (либо — масштаб, т. е. число знаков справа от десятичного разделителя)
	Десятичные значения — числовые значения, в которых используется десятичная точка (разделитель). В определении десятичных значений учитывается и сам разделитель

`REAL`
`DOUBLE PRECISION`
`FLOAT (p)`
`p` — точность
	Десятичные значения с плавающей точкой (float-point decimals) — десятичные значения, у которых точность и степень — переменной длины, они практически не имеют предела. Для этих значений допустимы любые точность и степень. `REAL` используется для определения данных в столбце обычной точности.
	`DOUBLE PRECISION` соответствует десятичным числам двойной точности. Значение считается значением обычной точности, если его точность задается числом от 1 до 21 включительно, двойной точности — если точность находится между 22 и 53 включительно. Для типа данных `FLOAT (p)` точность задается в скобках, например, `FLOAT (50)`

`DATETIME` представляется следующими типами:
`DATA`
`TIME (p)`
`TIMESTAMP (p)`
Состоит из следующих элементов: `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`
	Значения даты и времени:
	Дата
	Время
	Дата и время
	Состоит из следующих элементов: год, месяц, день, час, минута, секунда

Таблица 4.2 *(продолжение)*

Тип данных	Описание
`INTERVAL` где `p` — точность	Временной интервал. Следует отметить, что большинство реализаций SQL предлагают свои типы данных для хранения значений даты и времени, отличающиеся как по форме, так и по способу внутреннего представления хранимых данных. Для типов данных даты и времени длина обычно не задается

`"Value"`
`"34000"`
`"31 марта 2005"`
`38`
	Буквальные значения представляют собой последовательность символов, которая явно задается пользователем или программой. Буквальные значения могут быть любого типа, но предполагается, что значения известны.
	Для буквальных значений тип данных не объявляется — необходимо лишь указать нужную строку. Строки символов обычно требуют заключения в кавычки, числовые значения — нет.
	Буквальные значения используются в запросах к базам данных

`NULL` `' '` (пробел между одинарными кавычками)	Значения `NULL`. `NULL` используется для обозначения отсутствия значения. При наличии значения `NULL` поле выглядит как пустое. Поле со значением `NULL` представляется как не имеющее значения. Значение данного поля отличается от нулевого значения и от поля, содержащего только пробелы. (Таким образом, поля со значением `NULL` — это те поля, которые остаются пустыми при создании записи.)
`BOOLEAN`	Булевы величины (логические значения) могут принимать значения `TRUE` (истина), `FALSE` (ложь) или `NULL`

Таблица 4.2 *(окончание)*

Тип данных	Описание
Длинный текст	Многие СУБД могут хранить текстовые строки (длиной от 32 000 до 65 000 и более символов). Это удобно для хранения в базе целых документов. Обычно СУБД запрещает использовать столбцы с таким типом данных в интерактивных запросах
`IMAGE` (SQL-сервер)	Неструктурированные потоки байтов переменной длины. Обычно используются для хранения графических и видеоизображений, исполняемых файлов и других неструктурированных данных
Азиатские символы	В последнее время многие СУБД стали включать поддержку строк постоянной и переменной длины, содержащих символы азиатских алфавитов. Однако над такими строками, как правило, нельзя выполнять операции поиска и сортировки

Кроме типов данных, перечисленных в табл. 4.2, SQL позволяет задавать *пользовательский тип данных*, который может быть определен конкретным пользователем в зависимости от необходимости использования. Пользовательские типы данных строятся на основе имеющихся. Их можно использовать для определения атрибутов таблиц.

```
CREATE TYPE type_name AS OBJECT
  (COLUMN 1  DATA_TYPE [NULL | NOT NULL],
   ...
   COLUMN n  DATA_TYPE [NULL | NOT NULL]);
```

Пример. Создать пользовательский тип данных `PERSONAL`, который представляет собой объект, обладающий свойствами имя `NAME` (строка символов переменной длины, равной 30) и

табельный номер TAB_NUM (строка символов переменной длины, равной 8).

Решение.

```
CREATE TYPE PERSON_TYPE AS OBJECT
(NAME VARCHAR (30),
TAB_NUM VARCHAR (8));
```

Ссылка на данный пользовательский тип осуществляется следующим образом:

```
CREATE TABLE PERSONS
(PERSON PERSON_TYPE,
DATE_OF_BIRTH DATE);
```

Домены

Создание доменов:

```
CREATE DOMAIN domain_name
AS DATA_TYPE [NULL | NOT NULL];
```

Пример. Создать домен NUMBERS_DOMAIN, состоящий из действительных чисел.

Решение.

```
CREAT DOMAIN _NUMBERS AS REAL;
```

Ссылка на созданный домен:

```
CREATE TABLE TESTTABLE
   (TESTTABLE_ID NUMBER (9),
   NAME VARCHAR (30),
   TESTNUMBERS NUMBERS_DOMAIN);
```

Инструкция для обновления домена с добавлением ограничения:

```
ALTER DOMAIN domain_name
ADD CONSTRAINT domain_name_1
CHECK (VALUE условие);
```

Инструкция для удаления домена:

```
DROP DOMAIN domain_name CASCADE;
```

либо

```
DROP DOMAIN domain_name RESTRICT;
```

Константы

Константы, которые используются в языке SQL, представлены в табл. 4.3.

Таблица 4.3. *Наиболее употребительные константы SQL*

Константа	Описание

Целые и десятичные константы	Точные числовые литералы в инструкциях SQL представляются в виде обычных десятичных чисел с необязательным "+" или "–" перед ними:
	`28 - 658200.00 + 79600.873`
	В числовых константах нельзя ставить символы разделения разрядов между цифрами. Не все реализации баз данных допускают "+" перед числом. В некоторых СУБД при работе с денежными величинами можно ставить знак денежной единицы перед числом:
	`$ 0.87 $ 9800.00`

Константы с плавающей запятой	Приблизительные числовые литералы определяются с помощью символа `E` и имеют такой же формат, как и в большинстве языков программирования: `5.3 E3 0.91 E5`
Строковые константы	Заключаются в одинарные кавычки: `'Name'` Если необходимо включить в строковую константу одинарную кавычку, вместо нее следует поставить две одинарные кавычки либо `'\'`: `'I can''t'` или `'I can\'t'`, т. е. реально имеется строка: `"I can't"`

Таблица 4.3 *(окончание)*

Константа	Описание
Константы даты и времени	В реляционных СУБД значения даты, времени и интервалов времени представляются в виде строковых констант. Форматы этих констант различны в разных СУБД и меняются в зависимости от страны
Именованные константы	Специальные именованные константы SQL, возвращающие значения, хранимые в самой СУБД. Например, в ряде СУБД реализованы следующие константы: `CURRENT DATE` (текущая дата), `CURRENT TIME`, `CURRENT TIMESTAMP`, `USER`, `SESSION_USER`, `SYSTEM_USER`

Выражения

Выражения используются для выполнения операций над значениями, извлеченными из базы данных или используемыми для поиска в базе данных.

В соответствии со стандартом ANSI/ISO, в выражениях можно использовать четыре арифметические операции: +, –, *, /, например:

$$(X+Y),(X-Y),(X*Y),(X/Y),$$

где X и Y — это поля соответствующих таблиц (отношений), и другие значения.

Для формирования сложных выражений можно использовать скобки. Умножение и деление имеют более высокий приоритет, чем сложение и вычисление. Стандарт ANSI/ISO определяет, что преобразование целых чисел в десятичные и десятичных чисел в числа с плавающей запятой должно происходить автоматически.

Во многих СУБД допускается выполнение операций над датами и строками. Кроме того, наличие *встроенных функций* в

SQL позволяет выполнять различные операции преобразования и обработки данных, хранящихся в базе.

Рассмотрим наиболее употребительные функции языка SQL.

Функции для работы со строками

Соединение двух строк в одну осуществляется с помощью *функции конкатенации*. Синтаксис конкатенации, например, в MS SQL Server:

```
ИМЯ_СТРОКИ + ['']+] ИМЯ_СТРОКИ
```

Используя кавычки, в строку можно добавить почти любой символ. В некоторых реализациях SQL вместо одиночных кавычек могут использоваться двойные.

Функция `TRANSLATE` заменяет символы в строке символов в соответствии с указанным правилом замены. `TRANSLATE` имеет три аргумента: конвертируемую строку, список заменяемых символов и список символов, на которые заменяются символы из первого списка:

```
TRANSLATE (МНОЖЕСТВО_СТРОК, ЗНАЧЕНИЕ_1, ЗНАЧЕНИЕ_2)
```

Пример. Заменить все имеющиеся в строке символы следующим образом: K на O, L на P, M на R.

Решение.

```
SELECT CITY, TRANSLATE (CITY, 'KLM', 'OPR')
FROM TABLE;
```

Функция `REPLACE` используется для замены в строке некоторого символа (или строки символов) другим заданным символом (строкой символов):

```
REPLACE ('значение', 'значение', [NULL] 'значение')
```

Аналогична функции `TRANSLATE`, однако в данном случае заданный символ или строка заменяется другим заданным символом либо строкой.

Пример. Заменить все встречающиеся в строке буквы K на X.

Решение.

```
SELECT CITY,
REPLACE (CITY, 'K', 'X')
FROM TABLE;
```

Функция `UPPER` используется для изменения регистра символов заданной строки с нижнего на верхний.

Пример. Изменить регистр значений столбца `CITY` с нижнего на верхний.

Решение.

```
SELECT UPPER (CITY)
FROM TABLE;
```

Функция `LOWER` используется для изменения регистра символов заданной строки с верхнего на нижний.

Пример. Изменить регистр значений столбца `CITY` с верхнего на нижний.

Решение.

```
SELECT LOWER (CITY)
FROM TABLE;
```

Функция для выделения подстрок из строк: `SUBSTR (имя столбца, начальная позиция, длина)` — в Oracle; `SUBSTRING (имя столбца, начальная позиция, длина)` — в MS SQL Server.

Пример. Определить первые 3 символа строки `TAB_ID`.

Решение.

```
SELECT TAB_ID, SUBSTR (TAB_ID, 1, 3)
FROM TABLE;
```

Пример. Определить 5-й, 6-й и 7-й символы строки TAB_ID.

Решение.

```
SELECT TAB_ID,
SUBSTR (TAB_ID, 5, 3)
FROM TABLE;
```

Функция

```
INSTR (имя строки, 'множество символов',
[начальная позиция поиска, [, номер появления])
```

осуществляет *поиск заданного множества символов* в строке и возвращение позиции, в которой данное множество символов встретилось.

Пример. Определить для каждого названия из столбца PROD таблицы PRODUCT номер позиции, в которой первый раз встречается буква A.

Решение.

```
SELECT PROD, INSTR (PROD, 'A', 1, 1)
FROM PRODUCT;
```

Таким образом, данная инструкция возвращает для каждого названия столбца PROD номер позиции, в которой первый раз встречается буква A. Если символ A не будет найден в строке, то для позиции возвращается значение O.

Функция, которая *обрезает заданное множество символов* с начала строки, задается с использованием следующего синтаксиса:

```
LTRIM (строка символов, [, 'множество символов'])
```

Таким образом, функция LTRIM срезает в строке все символы слева до последнего символа искомой строки включительно.

Пример. Исключить (обрезать) символы 'LES' (а также 'ES', 'S') в начале всех имен столбца POS таблицы EMP_TBL.

Решение.

```
SELECT POS, LTRIM (POS, 'LES')
FROM EMP_TBL;
```

Функция *RTRIM* в отличие от функции LTRIM обрезает заданное множество символов в конце строки. Синтаксис RTRIM:

```
RTRIM (строка символов, [, 'множество символов'])
```

Пример. Исключить (обрезать) символы 'OR' в конце строки столбца POS таблицы EMP_TBL.

Решение.

```
SELECT POS, RTRIM (POS, 'OR')
FROM EMP_TBL;
```

Функция *DECODE* имеется не во всех реализациях SQL. DECODE используется для поиска строк по заданному значению или строке, и если строка найдена, в результатах запроса отображается другая заданная строка. Синтаксис данной функции:

```
DECODE (имя столбца, 'искомая 1', 'возвращаемая 1', [,
'искомая 2', 'возвращаемая 2', 'значение по умолчанию'])
```

Пример. Изменить в столбце CITY таблицы CITY_TBL значение MINSK на GRODNO, MOSCOW на SPB, остальным значением столбца присвоить значение KIEV.

Решение.

```
SELECT CITY, DECODE (CITY, 'MINSK', 'GRODNO', 'MOSCOW',
'SPB', 'KIEV')
FROM CITY_TBL;
```

Функция LENGTH (строка символов) определяет *длину строки* символов числа, значение даты или выражения в байтах.

Пример. Определить длину (общее число позиций) для каждого значения столбца PROD таблицы PROD_TBL.

Решение.

```
SELECT PROD, LENGTH (PROD)
FROM PROD_TBL;
```

Если необходимо добавить некоторые символы или пробелы в начало строки, то используется *функция* `LPAD`:

```
LPAD (множество строк, число, символ)
```

Пример. Добавить в начало каждого значения столбца `PROD` таблицы `PROD_TBL` необходимое количество точек таким образом, чтобы общее число символов в описании оказалось равным 25.

Решение.

```
SELECT LPAD (PROD, 25, '.')
FROM PROD_TBL;
```

Для добавления символа или пробела в конец строки используется *функция* `RPAD`:

```
RPAD (множество строк, число, символ)
```

Пример. Добавить в конец каждого значения столбца `PROD` таблицы `PROD_TBL` необходимое количество точек таким образом, чтобы общее число символов в описании оказалось равным 25.

Решение.

```
SELECT LPAD (PROD, 25, '.')
FROM PROD_TBL;
```

Функция `ASCII (строка)` возвращает *ASCII-код* самого левого символа в строке. Так, функция `ASCII ('A')` — возвратит 65, а функция `ASCII ('B')` — возвратит 66.

Математические функции

Общий вид математической функции можно записать следующим образом:

```
ФУНКЦИЯ (выражение)
```

Для всех реализаций SQL математические функции стандартны. С их помощью выполняются вычисления с числовыми значениями базы данных. Математические функции и вычисления не работают с символьными данными. Наиболее часто используются следующие функции:

- `ABS()` — вычисление абсолютного значения указанного числа;
- `ROUND()` — округление числа;
- `SQRT()` — вычисление квадратного корня;
- `SIGN()` — определение знака числа;
- `POWER()` — возведение в степень;
- `FLOOR()` и `CEIL()` — целая часть и ближайшее целое сверху;
- `EXP()` — вычисление экспоненты числа;
- `SIN()`, `COS()`, `TAN()` — вычисление различных тригонометрических функций.

Функции преобразования

Используются для преобразования одних типов данных в другие. Перечислим главные классы преобразования:

- символьного типа данных в числовой;
- числового типа в символьный;
- символьного типа в тип даты и времени;
- типа даты и времени в символьный.

По поводу имеющихся возможностей, правил преобразований и синтаксиса следует обращаться к документации для данного расширения реализации языка SQL.

Функции для работы с датами

Конкретные функции для работы с датами и временем зависят от используемого расширения языка SQL: их описание и возможности следует искать в соответствующем руководстве для конкретного сервера базы данных.

Следует отметить, что достаточно часто для сравнения дат и значений времени используется функция `OVERLAPS`. Она применяется для сравнения двух отрезков времени и возвращает `TRUE` (истина), если эти отрезки времени пересекаются, и `FALSE` (ложь) — в противном случае. Так, например, в результате сравнения двух временных отрезков:

```
(TIME '01:00:00', TIME '05:59:00')
OVERLAPS
(TIME '05:00:00', TIME '07:00:00')
```

будет возвращено значение `TRUE`, а в результате сравнения:

```
(TIME '02:00:00', TIME '06:59:00')
OVERLAPS
(TIME '07:00:00', TIME '09:00:00')
```

— значение `FALSE`.

Замечание

Достаточно часто необходимо конвертировать одни типы данных в другие. Реализация такой функции поддерживается, как правило, с использованием оператора `CAST`, синтаксис которого:

```
CAST (выражение AS новый тип данных)
```

Создание баз данных. Язык DDL

Как правило, за создание новых баз данных отвечает администратор крупной корпоративной сети. В СУБД, установленных на серверах более низкого уровня, некоторым поль-

зователям могут быть определены права для создания собственных баз данных, либо необходимые базы данных создаются централизованно, а пользователи затем работают с ними. В большинстве СУБД для создания и удаления баз данных используют инструкции `CREATE DATABASE` и `DROP DATABASE`.

Схемы в SQL

Под схемой базы данных в SQL понимается совокупность следующих объектов:

- таблиц вместе со связанными с ними структурами (столбцы, первичные и внешние ключи, ограничения и т. д.);
- представлений — виртуальных таблиц, создаваемых на основе реальных таблиц базы;
- доменов — пользовательских типов данных с дополнительными ограничениями на значения столбцов в таблицах схемы;
- утверждений — условий целостности базы данных, ограничивающих возможные сочетания данных из различных таблиц схемы;
- привилегий, предоставляемых пользователям на доступ к данным, их обновление, на модификацию структуры БД;
- наборов символов — структур, предназначенных для поддержки национальных языков и представления нелатинских символов (например, иероглифов);
- порядков сортировки, задающих правила сортировки текста для различных наборов символов;
- правил конвертирования текста, управляющих преобразованием текста из одного набора символов в другой и сравнением текстовых данных, использующих различные наборы символов.

Схемы базы данных создаются следующей инструкцией:

```
CREATE SCHEMA имя_схемы AUTHORIZATION имя_пользователя;
DEFAULT CHARACTERSET имя_набора_символов;
CREATE TABLE          (определение таблицы);
CREATE VIEW           (определение представления);
CREATE DOMAIN         (определение домена);
CREATE ASSERTION      (определение утверждения);
CREATE CHARACTERSET   (определение набора символов);
CREATE COLLATION      (определение сортировки);
CREATE TRANSLATION    (определение конвертирования);
GRANT                 (определение привилегий);
```

Пример. Создать для пользователя `LADA` схему базы данных `MYSCHEMA`, состоящую из двух таблиц `PERSONAL_DATA` (атрибуты `NAME` и `AGE`) и `CITY` (атрибуты `CITY_NAME` и `REGION`). Назначить полный доступ для всех пользователей для таблицы `PERSONAL_DATA` и запретить право на выборку данных из таблицы `CITY` для пользователя `IRINA`.

Решение.

```
CREATE SCHEMA MYSCHEMA AUTHORIZATION LADA;
CREATE TABLE PERSONAL_DATA
       (NAME VARCHAR (30),
        AGE INTEGER);
CREATE TABLE CITY
        (CITY_NAME VARCHAR (30),
         REGION VARCHAR (30));
GRANT ALL PRIVILEGES ON PERSONAL_DATA TO PUBLIC;
GRANT SELECT ON CITY NO IRINA;
```

Удаление схемы базы данных происходит с помощью инструкции:

```
DROP SCHEMA имя_схемы RESTRICT (CASCADE);
```

В данной инструкции следует обязательно задать правило удаления. Правило `CASCADE` позволяет все структуры в схеме удалять автоматически. Правило `RESTRICT` требует, чтобы пе-

ред удалением схемы из нее были удалены все вложенные структуры. Данное правило защищает схему базы данных от случайного удаления. Для изменения схемы используются обычные инструкции изменения входящих в нее объектов (например, `ALTER TABLE`).

Таблицы (отношения)

Создание таблицы

Таблицы в базе данных создаются с помощью инструкции `CREATE TABLE` (рис. 4.2).

Рассмотрим подробнее основные предложения инструкции `CREATE TABLE`.

Следует заметить, что, как правило, инструкция `CONSTRAINT` определяет ограничение. Это удобно использовать при описании внешних ключей со всеми налагаемыми условиями. В случае необходимости обращение к внешнему ключу будет происходить по *имени_ограничения*, подразумевая все описанные условия.

Замечание

Таблица, созданная инструкцией (см. рис. 4.2), изначально является пустой. Данные вносятся в нее в дальнейшем с помощью инструкции `INSERT`.

На схеме создания таблицы (рис. 4.2) представлены все инструкции, которые могут быть включены в определение таблицы. Однако реально некоторые инструкции из схемы `CREATE TABLE` могут отсутствовать.

Определение столбца

Определение столбца содержит следующую информацию:

- имя столбца;
- тип данных столбца;

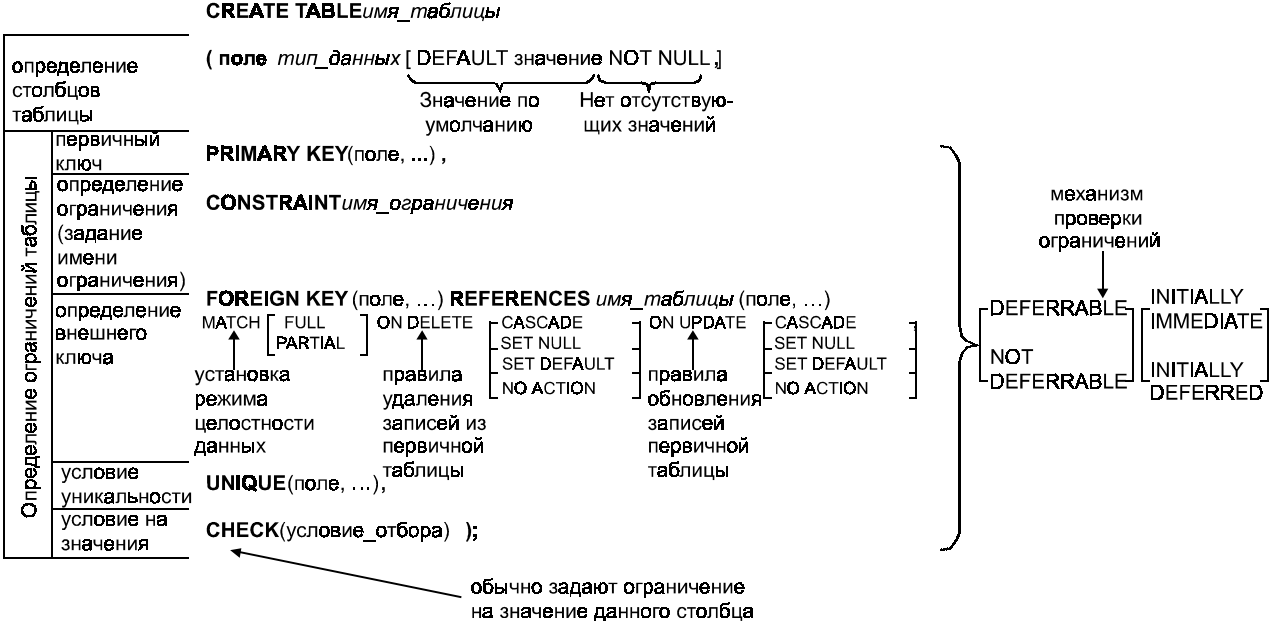


Рис. 4.2. Схема инструкции `CREATE TABLE`

- значение по умолчанию — может заноситься в столбец тогда, когда при внесении записей в таблицу не указано значение для данного столбца (в определении столбца может отсутствовать);
- указание на то, обязательно ли столбец должен содержать данные: `NOT NULL` предотвращает занесение в столбец значений `NULL` (может отсутствовать).

Определение первичного и внешнего ключей

Инструкцией `PRIMARY KEY` задается столбец или столбцы, которые образуют первичный ключ. СУБД автоматически следит за тем, чтобы первичный ключ был уникален. Кроме того, в определениях столбцов первичного ключа для значений должно быть указано `NOT NULL`.

В инструкции `FOREIGN KEY` задается внешний ключ таблицы, который определяет ее связь с другой таблицей. В инструкции указывается:

- столбец или столбцы создаваемой таблицы, которые образуют внешний ключ;
- таблица, связь с которой создает внешний ключ;
- имя столбца для внешней таблицы (может отсутствовать). Оно появляется только в сообщениях об ошибках;
- как СУБД должна обращаться со значениями `NULL` в одном или нескольких столбцах внешнего ключа (условия целостности); при создании таблицы можно установить один из двух режимов, обеспечивающих целостность данных:
  - `MATCH FULL` (полное соответствие) — требуется, чтобы внешние ключи были полностью равны первичному ключу связанной таблицы; в этом режиме ни одна часть внешнего ключа не может содержать значение `NULL`;
  - `MATCH PARTIAL` (частичное соответствие) — допускается, чтобы часть внешнего ключа имела значение `NULL` при условии, что остальная часть внешнего ключа равна со-

ответствующей части первичного ключа в связанной таблице;

- необязательное правило удаления для данного отношения ON DELETE:
  - RESTRICT — запрещает удалять строки из первичной таблицы (или NO ACTION);
  - CASCADE — при удалении строки из первичной таблицы автоматически удаляются все записи в связанной таблице;
  - SET NULL — при удалении записи из первичной таблицы всем внешним ключам в связанной таблице присваивается значение NULL;
  - SET DEFAULT — при удалении записи из первичной таблицы всем внешним ключам в связанной таблице присваивается определенное значение, по умолчанию установленное для данного столбца;
  - если ни одно из условий не указано, по умолчанию устанавливается правило NO ACTION (RESTRICT);
- необязательное правило обновления для данной таблицы ON UPDATE (см. правила удаления).

Условия уникальности

Условия уникальности обеспечивают уникальность значений для данного поля отношения: UNIQUE (поле).

Условия на значения

Условия на значения задают ограничения на значения данного столбца. Инструкция CHECK() аналогична условию отбора и возвращает значение TRUE или FALSE. Инструкция CHECK() может включать операторы сравнения, оператор BETWEEN...AND (проверка на принадлежность диапазону), LIKE, IN, а также составные условия отбора (с использованием операторов AND, OR и NOT).

Механизм проверки ограничений

Каждое создаваемое в базе данных ограничение любого типа может быть определено с одним из следующих атрибутов:

- `DEFERRABLE` — проверка может быть отложена до конца транзакции;
- `NOT DEFERRABLE` — проверка ограничения не может быть отложена до конца транзакции; обычно к этой категории относятся ограничения первичного ключа и условия уникальности, а также многие ограничения на значения столбцов. Такие ограничения должны проверяться после выполнения каждой инструкции, модифицирующей базу данных.

По умолчанию используется режим `NOT DEFERRABLE`.

Для создания ограничения, которое может быть отложено, необходимо включить в его объявление слово `DEFERRABLE` (однако следует помнить, что окончательное решение о проверках — откладывать или нет — определяется конкретной СУБД самостоятельно).

Для ограничения может быть определено начальное состояние:

- ограничение, определенное как `INTITALY IMMEDIATE`, проверяется после выполнения каждой инструкции;
- ограничение, определенное как `INITIALY DEFERRED`, проверяется только после окончания транзакции (этот атрибут не может использоваться с атрибутом `NOT DEFERRABLE`).

Начальное состояние ограничения вступает в силу сразу после его создания. Кроме того, ограничение переводится в это состояние перед началом каждой транзакции. `INITIALLY IMMEDIATE` обеспечивает более жесткий контроль целостности данных и используется по умолчанию. Если перед началом каждой транзакции необходимо перевести ограничение в отложенное состояние, то следует включить в ограничение `INITIALLY DEFERRED`.

SQL допускает динамическое изменение способа обработки ограничений с помощью инструкции `SET CONSTRAINTS`. Если

необходима срочная проверка какого-либо ограничения, то его можно вручную перевести в режим IMMEDIATE следующей инструкцией:

```
SET CONSTRAINTS имя_ограничения IMMEDIATE;
```

Один и тот же режим можно установить сразу и для нескольких ограничений:

```
SET CONSTRAINTS имя_ограничения 1, имя_ограничения 2,
      IMMEDIATE;
```

Режим обработки всех ограничений базы данных предусматривает следующая инструкция:

```
SET CONSTRAINTS ALL DEFERRED;
```

Пример. Записать инструкцию на SQL для создания таблицы STUDENTS (студенты) со следующими полями: STUDENT_ID (номер студента), SURNAME (фамилия), NAME (имя), _YEAR (курс), _GROUP (группа), CITY (город), FACULTY (факультет). Поле номер студента является первичным ключом таблицы STUDENTS. Значения поля YEAR находятся в пределах от 1 до 6. Значения поля _GROUP находятся в пределах от 1 до 20. По умолчанию полю CITY присваивается значение "Гродно". Значение поля FACULTY берется из следующего списка: {математический, физический, экономический, исторический, филологический, юридический, биологический}. Поля STUDENT_ID, SURNAME, NAME, _YEAR, _GROUP не могут принимать значений NULL.

Решение.

```
CREATE TABLE STUDENTS
              (STUDENT_ID INTEGER NOT NULL,
              SURNAME VARCHAR (30) NOT NULL,
              NAME VARCHAR (30) NOT NULL,
              _YEAR INTEGER NOT NULL
              CHECK (_YEAR BETWEEN 1 AND 6),
              _GROUP INTEGER NOT NULL
              CHECK (_GROUP BETWEEN 1 AND 20),
              CITY CHAR (15) DEFAULT 'Гродно',
              FACULTY CHAR (30)
              CHECK (FACULTY IN ('математический',
              'физический', 'экономический',
              'исторический', 'филологический',
              'юридический', 'биологический')),
              PRIMARY KEY (STUDENT_ID));
```

Пример. Записать инструкцию на SQL для создания таблицы SUBJECTS (предметы) со следующими полями: SUBJECT_ID (номер предмета), SUBJECT_NAME (название предмета), LECTURER (преподаватель), APPOINTMENT (должность преподавателя), SEMESTER (семестр), _HOURS (часы). Поле SUBJECT_ID является первичным ключом таблицы SUBJECTS. Поле SUBJECT_NAME не может принимать значение NULL. Значения поля SEMESTER находятся в пределах от 1 до 12. По умолчанию полю _HOURS присваивается значение "0.0". Значение поля APPOINTMENT берется из следующего списка: {ассистент, преподаватель, старший преподаватель, доцент, профессор}. Поля SUBJECT_ID, SEMESTER не могут принимать значений NULL.

Решение.

```
CREATE TABLE SUBJECTS
              (SUBJECT_ID INTEGER NOT NULL,
              SUBJECT_NAME VARCHAR (60) NOT NULL,
              LECTURER VARCHAR (30),
              APPOINTMENT CHAR (25)
              CHECK (APPOINTMENT IN ('ассистент',
              'преподаватель', 'старший преподаватель',
              'доцент', 'профессор')),
              SEMESTER INTEGER NOT NULL
              CHECK (SEMESTER BETWEEN 1 AND 12),
              _HOURS REAL DEFAULT 0.0,
              PRIMARY KEY (SUBJECT_ID));
```

Пример. Записать инструкцию на SQL для создания таблицы SESSION_RESULTS со следующими полями: STUDENT_ID (номер студента), SUBJECT_ID (номер предмета), EXAM_DATE (дата сдачи), MARK (оценка). Поля STUDENT_ID, SUBJECT_ID являются первичными ключами таблицы SESSION_RESULTS. Значения поля MARK находятся в пределах от 0 до 10; кроме того, по умолчанию принимается оценка 0. Поля STUDENT_ID, SUBJECT_ID не могут принимать значений NULL. Поле STUDENT_ID является внешним ключом к таблице STUDENTS, а поле SUBJECT_ID является внешним ключом к таблице SUBJECTS (названия связей определить произвольно). Для полей STUDENT_ID и SUBJECT_ID установить каскадные режимы обеспечения целостности для команд UPDATE и DELETE.

Решение.

```
CREATE TABLE SESSION_RESULTS
               (STUDENT_ID INTEGER NOT NULL,
               SUBJECT_ID INTEGER NOT NULL,
               EXAM_DATE DATE,
               MARK INTEGER DEFAULT 0
               CHECK (MARK BETWEEN 1 AND 10),
               CONSTRAINT SV PRIMARY KEY (STUDENT_ID,
               SUBJECT_ID),
               CONSTRAINT LEARN FOREIGN KEY (STUDENT_ID)
               REFERENCES STUDENTS ON UPDATE CASCADE ON
               DELETE CASCADE,
CONSTRAINT INCLUDES FOREIGN KEY (SUBJECT_ID) REFERENCES
SUBJECTS ON UPDATE CASCADE ON DELETE CASCADE);
```

Пример. Записать инструкцию на SQL для создания таблицы `STUDENT_1` со следующими полями: `STUDENT_ID` (номер студента), `SURNAME` (фамилия), `NAME` (имя), `_YEAR` (курс), `_GROUP` (группа), `CITY` (город), `FACULTY` (факультет), `THE_HEAD` (староста). Значение поля `THE_HEAD` должно быть идентификатором студента, являющегося старостой группы, в которой учится конкретный студент, для чего следует указать необходимые ограничения. Поле `STUDENT_ID` является первичным ключом таблицы `STUDENTS`. Значения поля `_YEAR` находятся в пределах от 1 до 6. Значения поля `_GROUP` находятся в пределах от 1 до 20. По умолчанию полю `CITY` присваивается значение `"Гродно"`, значение поля `FACULTY` берется из следующего списка: {математический, физический, экономический, исторический, филологический, юридический, биологический}. Поля `STUDENT_ID`, `SURNAME`, `NAME`, `_YEAR`, `_GROUP` не могут принимать значений `NULL`.

Решение.

```
CREATE TABLE STUDENT_1
               (STUDENT_ID INTEGER NOT NULL,
               SURNAME VARCHAR (30) NOT NULL,
               NAME VARCHAR (30) NOT NULL,
               _YEAR INTEGER NOT NULL
               CHECK (_YEAR BETWEEN 1 AND 6),
               _GROUP INTEGER NOT NULL
               CHECK (_GROUP BETWEEN 1 AND 20),
               CITY CHAR (15) DEFAULT 'Гродно',
```

```
FACULTY CHAR (30)
CHECK (FACULTY IN ('математический',
'физический', 'экономический',
'исторический', 'филологический',
'юридический', 'биологический')),
THE_HEAD INTEGER,
PRIMARY KEY (STUDENT_ID),
FOREIGN KEY (THE_HEAD) REFERENCES
STUDENT_1 (STUDENT_ID) ON UPDATE CASCADE
ON DELETE SET NULL);
```

Удаление таблицы

Удаление таблицы можно выполнить с помощью следующей инструкции:

```
DROP TABLE имя_таблицы [RESTRICT | CASCADE];
```

Если используется опция `RESTRICT` либо на таблицу ссылается ограничение или представление, то оператор `DROP` возвратит ошибку. При задании параметра `CASCADE` будет выполнено удаление не только самой таблицы, но и всех ссылающихся на таблицу представлений и ограничений.

Изменение определения таблицы (*ALTER TABLE*)

В процессе работы с таблицей часто необходимо провести различные модификации:

- добавление столбца в таблицу;
- удаление столбца из таблицы;
- изменение значения по умолчанию для какого-либо столбца;
- добавление либо удаление первичного ключа;
- добавление либо удаление внешнего ключа;
- добавление либо удаление условия уникальности;
- добавление либо удаление условия на значения.

Для проведения указанных модификаций таблицы могут использоваться следующие инструкции `ALTER TABLE`.

Добавление нового столбца происходит в соответствии с инструкцией:

```
ALTER TABLE имя таблицы ADD определение столбца;
```

Столбец добавляется в конец таблицы (справа). Предполагается, что он содержит NULL-значения. Чтобы задать конкретное значение по умолчанию для столбца, необходимо использовать инструкцию, содержащую соответствующие пояснения: [NOT NULL | DEFAULT значение].

Изменение столбца можно задать следующим образом:

```
ALTER TABLE имя таблицы ALTER имя столбца [SET DEFAULT |
DEFAULT] значение;
```

Для удаления столбца таблицы используется инструкция ALTER TABLE следующего вида:

```
ALTER TABLE имя таблицы DROP имя столбца [CASCADE |
RESTRICT];
```

Стандарт SQL требует, чтобы одна инструкция ALTER TABLE использовалась для удаления только одного столбца. Однако во многих СУБД данное ограничение снято. Операция удаления связана с целостностью данных:

- RESTRICT — если с удаляемым столбцом связан какой-либо объект в базе данных (внешний ключ, ограничение и т. д.), появится сообщение об ошибке и столбец не удалится;
- CASCADE — любой объект БД, связанный с удаляемым столбцом, также будет удален.

Добавление различных ограничений в таблицу можно произвести следующим образом:

```
ALTER TABLE имя таблицы ADD
        [определение_первичного_ключа |
        определение_внешнего_ключа |
        условие_уникальности |
        условие_на_значение];
```

Удаление ограничений в таблице происходит с помощью следующей инструкции:

```
ALTER TABLE имя таблицы DROP CONSRAINT имя ограничения
[CASCADE | RESTRICT];
```

Следует помнить, что удалить внешний ключ можно только тогда, когда создаваемая им связь имеет имя. Если имя присвоено не было, то задать эту связь в инструкции `ALTER TABLE` невозможно. В этом случае для удаления внешнего ключа необходимо удалить таблицу и воссоздать ее в новом формате.

Утверждения

Утверждение накладывает ограничение на содержимое базы данных и, как правило, связывает некоторой зависимостью столбцы нескольких таблиц. Утверждение задается в виде условия отбора с помощью оператора:

```
CREATE ASSERTION имя_утверждения
CHECK (описание ограничения)
```

Для удаления ограничения используют оператор:

```
DROP ASSERTION имя_ограничения
```

Для изменения определения утверждения необходимо его удалить и создать по-новому, т. к. в SQL нет инструкции `ALTER ASSERTION`.

Пример. Наложить на содержимое базы данных следующее ограничение: сумма заказов любого клиента не должна превышать его лимит кредита.

Решение.

```
CREATE ASSERTION CREDLIMIT
CHECK (CLIENTS.CREDIT_LIMIT >=
                      SELECT SUM (ORDERS.COST)
                      FROM ORDERS
                      WHERE
ORDERS.CLIENT_ID=CLIENTS.CLIENT_ID
                      GROUP BY ORDERS.CLIENT_ID)
```

Данное утверждение является частью определения базы данных, которое проверяется СУБД каждый раз, когда некоторая инструкция SQL пытается модифицировать таблицу CLIENTS или ORDERS.

Псевдонимы таблиц

Псевдоним — это имя, которое задает пользователь, оно заменяет имя некоторой таблицы. Псевдонимы используются, как правило, для замены длинных имен таблиц.

Псевдоним создается следующей инструкцией:

```
CREATE ALIAS имя_псевдонима
FOR полное_имя_таблицы;
```

Удаление псевдонима происходит с помощью команды DROP:

```
DROP ALIAS имя_псевдонима;
```

Псевдонимы используются в некоторых промышленных СУБД.

Индексы

Как отмечалось ранее, индекс представляет собой средство, которое обеспечивает быстрый доступ к строкам таблицы на основе одного или нескольких столбцов. Данные в индексе располагаются в убывающем или возрастающем порядке. СУБД использует индекс как предметный указатель: в индексе хранятся значения данных и указатели на строки, где эти данные встречаются.

СУБД просматривает индекс, чтобы найти требуемое значение, а затем с помощью указателя определяет нужную строку (строки) таблицы. Поиск в индексе осуществляется быстро, т. к. индекс отсортирован и его строки достаточно короткие.

Рекомендуется создавать индекс лишь для тех столбцов, которые часто используются в условиях отбора. Индексы удобны также в тех случаях, когда инструкции SELECT обращаются

к таблице чаще, чем инструкции INSERT и UPDATE. СУБД всегда создает индекс для первичного ключа.

К недостаткам индекса относится то, что он занимает на диске дополнительное место, а также есть необходимость обновления индекса каждый раз, когда в таблицу добавляется строка или обновляется индексный столбец таблицы. Это требует дополнительных затрат для выполнения инструкций INSERT и UPDATE, которые обращаются к данной таблице.

Индекс создается в соответствии со следующей инструкцией:

```
CREATE [UNIQUE] INDEX имя_индекса ON имя_таблицы
       (имя_столбца [ASC||DESC], ...);
```

При необходимости указываются значение уникальности (UNIQUE) и порядок сортировки (ASC — по возрастанию или DESC — по убыванию).

Во многих СУБД CREATE INDEX содержит дополнительные предложения, в которых задаются местоположение индекса на диске и рабочие параметры.

Удаление индекса происходит с помощью команды DROP:

```
DROP INDEX имя_индекса;
```

Представления

Представление — это запрос на выборку, которому присвоили имя и сохранили в базе данных.

Когда СУБД встречает в инструкции SQL ссылку на представление, происходит поиск его определения в базе данных. Далее осуществляется преобразование запроса с использованием представления в *эквивалентный* запрос к исходным таблицам, а затем — выполнения запроса. Таким образом, создается иллюзия существования представления в виде таблицы, в то же время сохраняется целостность исходных таблиц.

Если представление простое, то СУБД формируют каждую строку представления "мгновенно", извлекая данные из исходных таблиц. Если представление сложное, то СУБД *мате-*

риализует представление — СУБД выполняет запрос, определяющий представление, и сохраняет его результаты во временной таблице. Затем из этой таблицы берутся данные для пользовательского запроса. Когда временная таблица становится ненужной, она удаляется.

Перечислим преимущества использования представлений.

- *Безопасность.* Каждый пользователь имеет доступ к ограниченному числу представлений; таким образом, осуществляется ограничение доступа пользователей к хранимой информации.
- *Простота запросов.* Запрос к данным из многих таблиц можно организовать как запрос к данным из одной таблицы (представления).
- *Простота структуры.* Представления позволяют организовать каждому пользователю свою структуру базы данных, т. е. определять некоторое множество доступных виртуальных таблиц.
- *Защита от изменений.* Представление может возвращать непротиворечивый и неизменный образ структуры базы данных, даже если исходные таблицы разделяются, реструктурируются или переименовываются.
- *Целостность данных.* Если доступ к данным или ввод данных осуществляется с помощью представлений, СУБД может автоматически проверять, выполняются ли определенные условия целостности.

Однако наряду с преимуществами, следует помнить также и о некоторых недостатках использования представлений.

- *Производительность.* Представление создает лишь видимость существования таблицы. Если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением и на его выполнение может потребоваться много времени.
- *Ограничения на обновление.* При попытке обновления строк представления СУБД должна установить их соответствие

строкам исходных таблиц, а также обновить данные таблицы. Это можно сделать только для простых представлений; сложные представления обновить нельзя — они доступны только для выборки.

Создание представлений выполняется с помощью следующей инструкции:

```
CREATE VIEW имя_представления (имя_столбца_1, ...,
имя_столбца_n) AS запрос;
```

В скобках указаны столбцы, которые образуют заголовок представления.

Обновление представлений происходит лишь в том случае, когда СУБД может для каждой обновляемой строки представления найти исходную строку в исходной таблице, а для каждого обновляемого столбца представления — исходный столбец в исходной таблице.

Стандартом SQL сформулированы требования, разрешающие обновлять представления.

- Повторяющиеся строки не должны исключаться из таблицы результатов запроса.
- Должна быть задана только одна таблица, которую можно обновлять.
- Каждое имя в списке возвращаемых столбцов должно быть ссылкой на простой столбец; в этом списке не должны содержаться выражения, вычисляемые столбцы или статистические функции.
- Должны присутствовать только простые условия отбора для выполнения запроса. Не допускается использование группировок и задания условий для групп данных.

Удаление представлений осуществляется с использованием следующей команды:

```
DROP VIEW имя_представления (CASCADE || RESTRICT);
```

Параметр `CASCADE` означает, что СУБД должна удалить не только указанное представление, но и все представления, соз-

данные на его основе. Параметр RESTRICT означает, что удаление представления происходит лишь в том случае, если нет других представлений, созданных на его основе.

Другие объекты базы данных

Во многих СУБД команды CREATE, DROP, ALTER служат для образования дополнительных инструкций DDL, которые создают, удаляют и модифицируют другие объекты базы данных, имеющиеся в конкретной СУБД (триггеры, процедуры и т. д.).

Как правило, в различных СУБД используются следующие общие соглашения:

- первой в инструкции должна стоять команда CREATE, DROP, ALTER;
- второе слово определяет тип объекта;
- третье слово представляет собой имя объекта, подчиняющегося соглашениям об использовании имен в SQL.

Системный каталог

Системный каталог — совокупность системных таблиц, используемых СУБД для внутренних целей. Каждая таблица системного каталога содержит информацию об отдельном структурном элементе базы данных. Как правило, во всех реляционных СУБД содержатся следующие таблицы системного каталога:

- таблицы — описывается каждая таблица базы данных с указанием ее имени, владельца, числа содержащихся в ней столбцов, их размеров и т. д.;
- столбцы — описывается каждый столбец базы данных: имя столбца, имя таблицы, которой он принадлежит, тип данных столбца, его размер, разрешены ли значения NULL и т. д.;

- пользователи — в каталоге описывается каждый зарегистрированный пользователь базы данных: имя пользователя, пароль и некоторые другие данные;
- представления — описывается каждое представление, имеющееся в базе данных: указываются его имя, имя владельца, запрос, являющийся определением представления, и т. д.;
- привилегии — описывается каждый набор привилегий, предоставленных в базе данных: приводятся имена тех, кто предоставил привилегии, и тех, кому они предоставлены, указываются сами привилегии, объект, на который они распространяются, и т. д.

Как правило, каждая СУБД имеет свои имена системных таблиц. Запросы к системным каталогам разрешены почти во всех базах данных. Пользователи могут только извлекать информацию из системного каталога. СУБД запрещает модифицировать системные таблицы, т. к. это может нарушить целостность БД.

В СУБД можно дать *комментарии* (иногда до 254 символов) для каждой таблицы, преставления или столбца базы данных. Комментарии позволяют сохранить в системном каталоге краткое описание таблицы или других данных.

Комментарии также хранятся в системных таблицах, относящихся к таблицам и столбцам. Комментарии создаются инструкцией `COMMENT`:

```
COMMENT ON TABLE имя_таблицы IS  текст_комментария;
```

либо:

```
COMMENT ON COLUMN полное_имя_столбца IS
текст_комментария;
```

либо:

```
COMMENT ON имя_таблицы (имя_столбца_1 IS
текст_комментария, ...);
```

Манипуляция данными. Язык DML

Изменение содержимого базы данных включает три инструкции:

- `INSERT` — добавление новых строк в таблицу;
- `DELETE` — удаление строк из таблицы;
- `UPDATE` — обновление данных в таблице.

Добавление новых данных

Однострочная инструкция `INSERT` позволяет добавить в таблицу одну новую строку:

```
INSERT INTO имя_таблицы (имя_столбца_1, ...,
имя_столбца_n) VALUES (константа || NULL);
```

После имени таблицы в скобках указываются столбцы, в которые добавляются значения.

При добавлении в таблицу новой строки всем столбцам, имена которых отсутствуют в списке столбцов инструкции `INSERT`, автоматически присваивает значение `NULL`.

Для удобства в SQL разрешается не включать список столбцов в инструкцию `INSERT`. Если список столбцов опущен, он генерируется автоматически и в нем слева направо перечисляются все столбцы таблицы. Последовательность значений данных должна *точно* соответствовать порядку столбцов в таблице.

Многострочная инструкция INSERT добавляет в таблицу несколько строк, новым источником которых является запрос на выборку.

```
INSERT INTO имя_таблицы (имя_столбца_1, ...,
имя_столбца_n) ЗАПРОС;
```

Пример. Добавить в таблицу `OLD_ORDERS` (старые заказы) заказы, сделанные до 1 января 2005 года.

Решение.

```
INSERT INTO OLD_ORDERS (ORDER_ID, ORDER_DATE, COST)
  SELECT ORDER_ID, ORDER_DATE, COST
  FROM ORDERS
  WHERE ORDER_DATE < '01.01.2005'
```

Запрос, содержащийся внутри многострочной инструкции `INSERT`, должен удовлетворять следующим требованиям:

- в запрос нельзя включать `ORDER BY` (сортировка);
- таблица результатов запроса должна содержать количество столбцов, равное количеству столбцов в инструкции `INSERT`; кроме того, типы данных соответствующих столбцов таблицы результатов запроса и конечной таблицы должны быть совместными;
- запрос не может быть запросам на объединение (`UNION`) нескольких различных инструкций `SELECT`;
- имя целевой таблицы инструкции `INSERT` не может присутствовать в предложении `FROM` запроса на выборку или любого запроса, который является вложенным в него (запрет на добавление таблицы к самой себе).

Утилита пакетной загрузки (Import) служит для добавления в таблицу данных из внешнего файла. Она обычно используется для первоначального наполнения базы данных, а также для загрузки данных, либо содержащихся в другой компьютерной системе, либо собранных из различных источников.

Удаление данных

Инструкция `DELETE` удаляет выбранные строки из одной таблицы. В предложении `WHERE` указывается критерий отбора строк, которые должны быть удалены.

```
DELETE FROM имя_таблицы [WHERE условие_отбора];
```

Пример. Удалить информацию из базы данных о служащем Петрошенко.

Решение.

```
DELETE FROM EMPLOYEE
WHERE NAME = 'Petroshenko';
```

Пример. Удалить данные о клиентах, чьи идентификаторы 48,173, 214.

Решение.

```
DELETE FROM EMPLOYEE
WHERE EMP_ID IN (48, 173, 214);
```

Пример *(инструкция* `DELETE` *с подчиненным запросом)*. Необходимо удалить все заказы, которые оформлены служащим Петровым.

Решение.

Для определения всех заказов, которые оформил Петров, необходимо знать его идентификатор, т. е. нужно использовать две таблицы `ORDERS` и `EMPLOYEE`.

```
DELETE FROM ORDERS
WHERE EMP = (SELECT EMP_ID FROM EMPLOYEE WHERE SURNAME =
'Petroff')
```

Для удаления *всех строк в таблице* следует использовать инструкцию:

```
DELETE FROM имя_таблицы;
```

Обновление данных

Инструкция `UPDATE` обновляет значения одного или нескольких столбцов в выбранных строках одной таблицы.

```
UPDATE имя_таблицы SET (имя_столбца_1 = выражение_1, ...,
имя_столбца_n = выражение_n) [WHERE условие_отбора];
```

Пример. Перевести всех рабочих из первого цеха (идентификатор 11) в третий (идентификатор 33) и уменьшить объемы выполняемой ими работы на 7%.

Решение.

```
UPDATE EMPLOYEE
SET WORKSHOP = 33, VOLUME = 0.93 * VOLUME
WHERE WORKSHOP = 11;
```

Предложение `SET` в инструкции `UPDATE` представляет собой список операций присваивания, отделяемых друг от друга запятыми. Каждый целевой столбец должен встречаться в списке только один раз; не должно быть двух операций присваивания для одного и того же целевого столбца. Выражение в операции присваивания может быть любым доступным SQL-выражением, результирующее значение которого имеет тип данных, соответствующий целевому столбцу.

Если выражение в операции присваивания содержит ссылку на один из столбцов целевой таблицы, то для вычисления используется значение этого столбца в текущей строке, которое было до обновления. То же самое справедливо для ссылок на столбцы в предложении `WHERE`.

Обновление всех строк происходит согласно следующей инструкции:

```
UPDATE имя_таблицы SET (имя_столбца_1 = выражение_1, ...,
имя_столбца_n = выражение_n);
```

Инструкция `UPDATE` с подчиненным запросом аналогична соответствующей инструкции `DELETE`.

Пример. Увеличить кредит на 300 000 руб. для тех клиентов, которые сделали заказ на сумму более 20 000 руб.

Решение.

```
UPDATE CLIENTS
SET CREDIT = CREDIT + 300000.00
WHERE CLIENT_ID IN (SELECT DISTINCT CUSTOMER
FROM ORDERS
WHERE COST > 20000.00);
```

Запросы на выборку данных. Язык DQL

Инструкция *SELECT* для выборки данных

Для извлечения информации из базы данных и представления ее в виде таблицы данных с требуемыми столбцами используется инструкция `SELECT`, синтаксис которой приведен далее.

```
SELECT [ALL||DISTINCT] список столбцов запроса
FROM список таблиц, содержащих данные
WHERE условие отбора строк
GROUP BY список столбцов запроса, по которым происходит группировка
HAVING условие отбора групп, полученных с помощью GROUP BY
ORDER BY столбцы запроса (либо их номера), подлежащие сортировке [ASC||DESC];
```

Рассмотрим подробнее каждое предложение инструкции на выборку данных.

Предложение *SELECT*

В предложении `SELECT` указывается список столбцов, которые должны быть получены. Столбцы отделяются друг от друга запятыми. Столбцы в таблице результатов будут расположены в том же порядке, что и элементы списка возвращаемых столбцов. Возвращаемый столбец может представлять собой:

- имя столбца, идентифицирующее один из столбцов, содержащихся в таблицах, которые перечислены в предложении `FROM`;
- константу, показывающую в каждой строке запроса одно и то же значение;
- выражение, содержащее вычисляемое значение, помещаемое в таблицу результатов запроса.

Если в списке возвращаемых столбов запроса на выборку указать первичный ключ таблицы, то каждая строка запроса будет уникальной. В противном случае результаты запроса могут содержать повторяющиеся строки.

Повторяющиеся строки из таблицы результатов запроса можно удалить, если в инструкции SELECT перед списком возвращаемых столбцов указать предикат DISTINCT. Если предикат DISTINCT не указан, то повторяющиеся строки не удаляются и считается, что ALL используется по умолчанию.

Столбцам можно присвоить псевдонимы с помощью предиката AS:

```
SELECT имя_столбца AS новое_имя_столбца
```

Предложение *FROM*

В предложении FROM указывается список таблиц, которые содержат элементы данных, извлекаемые запросом. Таблицы отделяются друг от друга запятыми. При необходимости (построение рекурсивных запросов, длинные имена таблиц, полные имена таблиц) в предложении FROM можно задать псевдоним таблицы (новое имя таблицы):

```
FROM имя_таблицы новое_имя_таблицы (псевдоним)
```

Данное имя можно использовать в инструкции запроса в качестве непосредственной ссылки на столбцы этой таблицы.

Предложение *WHERE*

Предложение WHERE указывает на то, какие строки следует включать в результаты запроса. Для отбора строк, включаемых в результаты запроса, используется соответствующее условие отбора. Существуют пять основных условий отбора (предикатов): сравнение, проверка на принадлежность диапазону, проверка на членство во множестве, проверка на соответствие шаблону и проверка на равенство значению NULL.

Сравнение

Синтаксис сравнения можно условно записать следующим образом:

```
выражение_1 [ = | <> | < | <= | > | >= ] выражение_2
```

Значение одного выражения сравнивается со значением другого выражения (для каждой строки данных). Выражения могут быть как простыми (например, содержать одно имя столбца или константу), так и сложными (содержать арифметические операции, функции).

Когда СУБД сравнивает значения двух выражений, могут получиться следующие результаты:

- если сравнение истинно, то результат проверки имеет значение `TRUE`;
- если сравнение ложно, то результат проверки имеет значение `FALSE`;
- если хотя бы одно из двух выражений имеет значение `NULL`, то результатом проверки будет `NULL`.

При определении условия отбора всегда необходимо помнить об обработке значений `NULL`, т. к. часто строки, содержащие `NULL`-значения, могут "исчезать" при выполнении запроса.

Следует помнить, что в трехзначной логике SQL условие отбора может иметь значения `TRUE`, `FALSE` или `NULL`, а в результате запроса попадают только те строки, для которых условие отбора равно `TRUE`.

Проверка на принадлежность диапазону

Синтаксис проверки на принадлежность диапазону:

```
проверяемое выражение [ | NOT ] BETWEEN нижнее_выражение
AND верхнее_выражение;
```

Оператор `BETWEEN...AND` проверяет, находится ли элемент данных между двумя заданными значениями. В условие отбора входят три выражения. Первое выражение определяет проверяемое значение; второе и третье выражение определяют

верхний и нижний пределы проверяемого диапазона. Типы данных трех выражений должны быть сравнимыми.

Проверяемое выражение, заданное в операторе BETWEEN...AND, может быть любым допустимым выражением, однако обычно оно представляет собой короткое имя столбца.

Правила обработки значений NULL в проверке BETWEEN:

- если проверяемое выражение имеет значение NULL либо если оба выражения, определяющие диапазон, равны NULL, то проверка BETWEEN возвращает NULL;
- если выражение, определяющее нижнюю границу диапазона, имеет значение NULL, то проверка BETWEEN возвращает FALSE, когда проверяемое значение больше верхней границы диапазона, и NULL в противном случае;
- если выражение, определяющее верхнюю границу диапазона, имеет значение NULL, то проверка BETWEEN возвращает FALSE, когда проверяемое значение меньше нижней границы диапазона, и NULL в противном случае.

Следует помнить, что проверка на принадлежность диапазону не расширяет возможности SQL, т. к. ее можно выразить в виде двух операций сравнения.

Проверка на членство во множестве

Синтаксис проверки на членство во множестве:

```
проверяемое_выражение [ | NOT ] IN список_констант
```

В данном случае проверяется, совпадает ли значение выражения с одним из значений заданного множества.

С помощью проверки NOT IN можно убедиться в том, что элемент данных не является членом заданного множества. Проверяемое выражение в операторе IN может быть любым допустимым выражением, однако обычно оно представляет собой короткое имя столбца. Если результатом проверяемого выражения является значение NULL, то проверка IN также возвращает NULL. Все элементы в списке заданных значений

должны иметь один и тот же тип данных, который должен быть сравним с типом данных проверяемого выражения.

Проверку IN также можно выразить через проверку на сравнение.

Следует избегать множеств, содержащих один элемент. Например,

```
CITY IN ('Minsk')
```

лучше заменить простым сравнением:

```
CITY = 'Minsk'
```

Проверка на соответствие шаблону

Синтаксис проверки на соответствие шаблону:

```
имя_столбца [ | NOT ] LIKE шаблон
                                [ | ESCAPE символ_пропуска]
```

Проверяется, соответствует ли строковое значение, содержащееся в столбце, определенному шаблону. ***Шаблон*** представляет собой строку, в которую может входить один или более ***подстановочных знаков***, которые интерпретируются особым образом.

Подстановочный знак * (либо %) используется для поиска любой последовательности символов.

Подстановочный знак ? (либо символ подчеркивания) используется для поиска отдельного фиксированного символа.

Подстановочные знаки можно помещать в любое место строки шаблона, и в одной строке может содержаться несколько подстановочных знаков.

При проверке строк на соответствие шаблону может оказаться, что подставочные знаки входят в строку символов в качестве литералов (например, тот же знак *). Для проверки наличия в строке литералов, использующихся в качестве подставочных знаков, используются символы пропуска. Если в шаблоне встречается такой символ, то символ, следующий непосредственно за ним, считается не подставочным знаком, а литералом (происходит пропуск символа).

Например, найти продукты, код которых начинается с четырех букв "A*DC".

```
SELECT PRODUCT_ID
FROM PRODUCTS
WHERE PRODUCT_ID LIKE 'A$*DC*' ESCAPE '$'.
```

Здесь: первый символ * в шаблоне, следующий за символом пропуска $, считается литералом, второй — подставочным знаком.

Проверка на равенство значению *NULL*

Синтаксис проверки на равенство значению NULL:

```
имя_столбца IS [ | NOT ] NULL
```

Проверяется, содержится ли в столбце значение NULL. Значения NULL обеспечивают возможность применения трехзначной логики в условиях отбора. Для любой заданной строки результат применения условия отбора может быть TRUE, FALSE или NULL (когда в одном из столбцов содержится значение NULL). Часто бывает необходимо явно проверять значения столбцов на равенство NULL и непосредственно обрабатывать их. Для этого имеется специальное предложение IS NULL.

Предикат IS NOT NULL позволяет отыскать строки, которые не содержат значений NULL.

Проверка на NULL не может возвратить значение NULL в качестве результата. Она всегда возвращает TRUE или FALSE.

Нельзя проверить значение на равенство NULL с помощью операции сравнения (равенства).

Условия отбора могут также быть составными, в которые включены различные ограничения, записанные с использованием соответствующих операторов.

Оператор OR используется для объединения двух условий отбора, из которых как минимум одно должно быть истинным (табл. 4.4).

Таблица 4.4. *Таблица истинности оператора* `OR`

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Оператор `AND` используется для объединения двух условий отбора, оба из которых должны быть истинными (табл. 4.5).

Таблица 4.5. *Таблица истинности оператора* `AND`

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Оператор `NOT` используется для отбора строк, для которых условие отбора ложно (табл. 4.6).

Таблица 4.6. *Таблица истинности оператора* `NOT`

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Составные условия отбора

Составные условия отбора представляют собой объединение с помощью операторов `AND`, `OR` и `NOT` более двух условий отбора, которые можно использовать в предложениях `WHERE` или `HAVIHG`:

```
WHERE [ | NOT ] составное_условие_отбора_строк
HAVING [ | NOT ] составное_условие_отбора_групп
```

Оператор NOT имеет наивысший приоритет, далее идет оператор AND, затем OR. Рекомендуется использовать круглые скобки при составлении сложных условий отбора.

Проверка *IS*

Оператор IS проверяет значение результата логического выражения. Например:

```
((COST - QUOTA) > 53000.00) IS UNKNOWN
```

используют, чтобы отыскать строки, в которых значение COST если и превышает значение QUOTA, то незначительно.

Проверку IS можно переписать также в виде:

```
NOT ((COST - QUOTA) > 53000.00)
```

Рекомендуется по возможности использовать AND, OR и NOT. Однако не всегда удается избежать проверки IS UNKNOWN.

Статистические функции

Для вывода обобщающей информации, содержащейся в базе, используются итоговые запросы на выборку. Итоговый запрос добавляет в инструкцию SELECT следующие элементы:

- статистические функции (табл. 4.7), использующие в качестве аргумента какой-либо столбец или выражение в инструкции SELECT;
- предложение GROUP BY;
- предложение HAVING.

Таблица 4.7. *Статистические функции*

Статистическая функция	Описание
SUM (выражение || DISTINCT имя_столбца)	Вычисление суммы всех значений столбца
AVG (выражение || DISTINCT имя_столбца)	Вычисление среднего всех значений столбца

Таблица 4.7 *(окончание)*

Статистическая функция	Описание
`MIN (выражение)`	Нахождение наименьшего среди всех значений столбца
`MAX (выражение)`	Нахождение наибольшего среди всех значений столбца
`COUNT (DISTINCT имя_столбца)`	Подсчет количества значений, содержащихся в столбце
`COUNT (*)`	Нахождение количества строк в таблице результатов запроса

Статистические функции используют в качестве аргумента какой-либо столбец данных целиком, а возвращают одно значение, которое является итогом для данного столбца. В списке возвращаемых столбцов вместо имени любого столбца можно указывать статистическую функцию. Статистическая функция не может быть аргументом для другой функции такого же типа, т. е. нельзя допускать вложений статистических функций.

В стандарте ANSI/ISO определены следующие правила обработки значений `NULL` в статистических функциях:

- значения столбца, равные `NULL`, при вычислении результата функции исключаются;
- если все значения в столбце равны `NULL`, то функции `SUM()`, `AVG()`, `MIN()` и `MAX()` возвращают значение `NULL`; функция `COUNT()` возвращает ноль;
- если в столбце нет значений (столбец пустой), то функции `SUM()`, `AVG()`, `MIN()` и `MAX()` возвращают значение `NULL`; функция `COUNT()` возвращает ноль;
- функция `COUNT(*)` подсчитывает количество строк и не зависит от наличия или отсутствия в столбце значений `NULL`; если строк в таблице нет, то эта функция возвращает ноль.

Условие DISTINCT указывается, как правило, в начале списка возвращаемых столбцов (в предложении SELECT) и служит для удаления повторяющихся строк из таблицы результатов запроса. При использовании DISTINCT в статистической функции ее аргументом должно быть простое имя столбца — аргумент не может быть выражением. В одном запросе (в одной инструкции SELECT) DISTINCT употребляется только один раз.

Пример. Найти среднюю стоимость заказов, общую стоимость заказов, а также среднюю стоимость заказов в процентах от лимитов кредита клиентов.

Решение.

```
SELECT AVG(COST), SUM(COST), (100*AVG(COST /
CREDIT_LIMIT))
FROM ORDERS, CLIENTS
WHERE ORDERS CUSTOMER=CLIENTS.CLIENT_ID;
```

Предложение *GROUP BY*

Предложение GROUP BY позволяет создать итоговый запрос. Обычный запрос включает в результаты запроса по одной записи для каждой строки таблицы. Итоговый запрос вначале группирует строки базы данных по определенному признаку, а затем включает в результаты запроса одну итоговую строку для каждой группы. Столбцы, указанные в предложении GROUP BY, называются *столбцами группировки*, т. к. они определяют признак, по которому строки делятся на группы.

Если в запросе с группировкой присутствует статистическая функция, то она применяется по отдельности к каждой группе и выдает результат в виде строки для каждой группы.

В SQL можно группировать результаты запроса на основании двух или более столбцов. Однако запрос возвращает одну итоговую строку для соответствующих столбцов, участвующих в группировке. В SQL нельзя создать группы и подгруппы, например, с двумя уровнями итоговых результатов.

Столбцы с группировкой должны представлять реальные столбцы таблиц, которые перечислены в предложении FROM, т. к. нельзя группировать строки на основании значения вычисляемого выражения. Кроме того, если две строки имеют значение NULL в одинаковых столбцах группировки и идентичные значения во всех остальных столбцах группировки, они помещаются в одну группу.

Предложение *HAVING*

Предложение HAVING применяется для отбора групп строк и по аналогии с предложением WHERE также может содержать условие отбора. В условие отбора, как правило, входят:

- константа;
- статистическая функция, возвращающая одно значение для всех строк, входящих в группу;
- столбец группировки, который по определению имеет одно и то же значение во всех строках группы;
- выражение, включающее в себя все перечисленные выше элементы.

Правила обработки значений NULL в условиях отбора для предложения HAVING аналогичны условиям отбора предложения WHERE.

Предложение HAVING почти всегда используется с GROUP BY. Если HAVING применяется без GROUP BY, то СУБД рассматривает полные результаты запроса как одну группу, состоящую из всех строк.

Предложение *ORDER BY*

Предложение ORDER BY сортирует результаты запроса на основании данных, содержащихся в одном или нескольких столбцах. Если это предложение не указано, результаты запроса не будут отсортированы.

В предложении ORDER BY первый столбец является *главным* ключом сортировки; столбцы, следующие за ним — *второ-*

степенные ключи. Можно сортировать результаты запроса по любому элементу списка возвращаемых столбцов. По умолчанию данные сортируются в порядке возрастания (либо указать ASC). Для сортировки в порядке убывания следует включить в предложение сортировки слово DESC.

Объединения в многотабличных запросах на выборку

В запросах может участвовать любое количество таблиц, хранящихся в базе данных. Таблица, получающаяся в результате формирования строк и содержащая данные изо всех таблиц предложения FROM, называется *объединением* таблиц. К особенностям многотабличных запросов относятся:

- возможное использование полных имен столбцов для исключения неоднозначных ссылок на столбцы; таблица, заданная в полном имени столбца, должна соответствовать одной из таблиц, указанных в инструкции FROM;
- предложение SELECT * позволяет в многотабличном запросе вывести все столбцы таблиц, указанных в предложении FROM.

Объединение результатов нескольких запросов (операция *UNION*)

Объединить результаты нескольких запросов можно с использованием операции UNION. Запросы, которые объединяются операцией UNION, должны удовлетворять определенным условиям:

- таблицы должны содержать одинаковое число столбцов;
- тип данных каждого столбца таблицы должен совпадать с соответствующим типом данных из столбца другой таблицы;

- ни одна из таблиц не может быть отсортирована с помощью инструкции ORDER BY; однако объединенные результаты запроса можно отсортировать;
- имена столбцов в объединяемых запросах не обязательно должны быть одинаковыми, т. к. столбцы результатов запроса, возвращенные UNION, являются безымянными;
- в операции SELECT, участвующей в операции UNION, запрещается использовать выражения;
- операция UNION по умолчанию удаляет все повторяющиеся строки. Чтобы их оставить, необходимо задать инструкцию ALL;
- объединенные результаты запросов, возвращенные операцией UNION, можно отсортировать с помощью инструкции ORDER BY, следующей за последней инструкцией SELECT;
- операция UNION может быть использована многократно, чтобы объединить результаты трех и более запросов. Чтобы указать последовательность выполнения в запросах на объединение, всегда следует использовать круглые скобки.

Объединение по равенству

Объединение по равенству основано на связях между таблицами (с помощью первичных и внешних ключей). В SQL не требуется, чтобы связанные столбцы включались в результаты многотабличного запроса. Достаточно указать равенство по ключевым полям в инструкции WHERE. В условиях отбора многотабличного запроса кроме связанных столбцов можно также задавать и другие условия отбора, чтобы больше сузить результаты запроса. Кроме того, любые два столбца из двух таблиц могут быть связанными, если они имеют сравнимые типы данных. Объединение таблиц по равенству — это декартово произведение таблиц, состоящее из всех возможных пар строк таблиц.

Объединение по неравенству

Объединение по неравенству — это объединение таблиц на основе операций сравнения, которое не включает в условие WHERE связи между таблицами. Объединения по неравенству применяются редко. Как правило, они используются в приложениях, предназначенных для поддержки принятия решений.

Рекурсивное объединение (самообъединение)

Рекурсивное объединение (самообъединение) таблицы основано на использовании "предполагаемой копии" этой же таблицы — *псевдонима таблицы*. Для создания псевдонима таблицы в предложении FROM для каждой копии таблицы назначается свой псевдоним, который ставится после реального имени таблицы.

Внутреннее объединение (*INNER JOIN*)

Внутреннее объединение (INNER JOIN) определяется по умолчанию и представляет собой подмножество декартова произведения, образующееся путем удаления тех строк таблицы, которые не удовлетворяют условию отбора. Внутреннее объединение рассматривает только взаимосвязанные строки таблиц.

Перекрестное объединение (*CROSS JOIN*)

Перекрестное объединение (CROSS JOIN) возвращает таблицу $m \times n$ строк, состоящую из всех возможных пар (декартово произведение) строк обеих таблиц.

Полное внешнее объединение (*FULL JOIN*)

Полное внешнее объединение (`FULL JOIN`) возвращает таблицу, содержащую все строки внутреннего объединения таблиц, расширенные значениями `NULL` (в том случае, если в связанной таблице отсутствуют соответствующие значения).

Полное внешнее объединение строится следующим образом:

1. Создать внутреннее объединение таблиц.
2. Каждую строку первой таблицы, которая не имеет связи ни с одной строкой второй таблицы, добавить в результаты запроса, присваивая всем столбцам второй таблицы значение `NULL`.
3. Каждую строку второй таблицы, которая не имеет связи ни с одной строкой первой таблицы, добавить в результаты запроса, присваивая всем столбцам первой таблицы значение `NULL`.
4. Результирующая таблица является внешним объединением двух таблиц.

Левое внешнее объединение (*LEFT JION*)

Левое внешнее объединение (`LEFT JION`) возвращает таблицу, содержащую все строки внутреннего объединения, расширенные значениями `NULL` строки первой (левой) таблицы предложения `FROM`, не удовлетворяющие условию отбора. Левое внешнее объединение получается, если выполнить пункты 1 и 2 из полного внешнего объединения.

Правовое внешнее объединение (*RIGHT JION*)

Правовое внешнее объединение (`RIGHT JION`) возвращает таблицу, содержащую все строки внутреннего объединения, расширенные значениями `NULL` строки второй таблицы, не удовлетворяющие условию отбора. Правое внешнее объединение

получается, если выполнить пункты 1 и 3 из полного внешнего объединения.

Расширенный запрос на объединение (*UNION JOIN*)

Расширенный запрос на объединение (`UNION JOIN`) возвращает таблицу, содержащую все строки первой и второй таблиц, расширенные значениями `NULL`. Результаты расширенного запроса на объединение можно получить, если выполнить пункты 2 и 3 из полного внешнего объединения.

Задание объединений

Рассмотренные объединения таблиц можно задавать в предложении `FROM`:

```
FROM таблица_1 INNER || FULL || LEFT || RIGHT || CROSS ||
UNION JOIN таблица_2
```

Замечание

Следует помнить, что включение в предложение `FROM` какого-либо объединения (исключение — `INNER`, которое используется по умолчанию), связывающего более двух таблиц, влечет за собой выполнение декартова произведения таблиц, результат которого зависит от порядка выполнения операции объединения. В силу этого необходимо указывать приоритет декартова произведения путем расстановки скобок в предложении `FROM`.

В конкретных СУБД система записи различных объединений может быть перенесена в предложение `WHERE` (см. например, табл. 4.8).

Таблица 4.8. *Пример записи объединений в предложении* `WHERE` *для SQL-Server*

Вид записи	Описание
`WHERE столбец_1 *=* столбец_2`	Полное внешнее объединение
`WHERE столбец_1 *= столбец_2`	Левое внешнее объединение

Таблица 4.8 *(окончание)*

Вид записи	Описание
`WHERE столбец_1 =* столбец_2`	Правое внешнее объединение
`WHERE столбец_1 *>= столбец_2`	Использование с другими операторами сравнения

Правила выполнения запроса на выборку

Правила выполнения запроса на выборку.

1. Если запрос представляет собой запрос на объединение (`UNION`) инструкций `SELECT`, для каждой из этих инструкций выполнить действия 2—7 и получить отдельную таблицу результатов.
2. Сформировать произведение таблиц, перечисленных в предложении `FROM`. Если в предложении `FROM` указана только одна таблица, то произведением будет она сама.
3. Если имеется предложение `WHERE`, то применить заданное в нем условие отбора к каждой строке таблицы произведения и оставить в ней только те строки, для которых это условие выполняется — `TRUE` (группы строк, для которых условие отбора имеет значение `FALSE` или `NULL`, отбрасываются).
4. Если имеется предложение `GROUP BY`, разделить строки, оставшиеся в таблице произведения, на группы таким образом, чтобы строки в каждой группе имели одинаковые значения во всех столбцах группировки.
5. Если имеется предложение `HAVING`, применить заданное в нем условие отбора к каждой группе строк и оставить в таблице произведения только те группы, для которых это условие выполняется — `TRUE` (группы строк, для которых условие отбора имеет значение `FALSE` или `NULL`, отбрасываются).

6. Для каждой из оставшихся строк (или для каждой группы строк) вычислить значение каждого элемента в списке возвращаемых столбцов и создать одну строку в таблице результатов запроса. При любой ссылке на столбец берется значение столбца для текущей строки (или группы строк). В качестве аргумента статической функции используются значения столбца из всех строк, входящих в группу, если указано предложение GROUP BY; в противном случае используются значения столбца из всех строк таблицы результатов.
7. Если указан предикат DISTINCT, удалить из таблицы результатов запроса все повторяющиеся строки.
8. Если запрос является запросом на объединение (UNION) инструкции SELECT, объединить результаты выполнения отдельных инструкций в одну таблицу результатов запроса. Удалить из нее повторяющиеся строки, если не указан предикат ALL.
9. Если имеется предложение ORDER BY, отсортировать результаты запроса.

Некоторые замечания о подчиненных запросах

Подчиненный запрос представляет собой запрос, результаты которого используются в качестве составной части другого запроса. Подчиненные запросы содержатся в предложении WHERE или HAVING.

Если подчиненный запрос содержится в предложении WHERE, его результаты используются для отбора отдельных строк, данные из которых заносятся в таблицу результатов запроса. Когда подчиненный запрос содержится в предложении HAVING, его результаты используются для отбора групп строк, данные из которых заносятся в таблицу результатов запроса.

Отличия подчиненного запроса от инструкции SELECT.

- Таблица результатов подчиненного запроса всегда состоит из одного столбца, т. е. в предложении SELECT подчиненно-

го запроса всегда указывается только один возвращаемый столбец.

- В подчиненный запрос не входит предложение `ORDER BY`. Результаты подчиненного запроса используются только внутри главного запроса, поэтому нет необходимости в их сортировке.
- Имена столбцов в подчиненном запросе могут являться ссылками на столбцы таблиц главного запроса. Такие *внешние ссылки* связывают подчиненный запрос с "текущей" строкой данного запроса.
- Подчиненный запрос не может быть запросом на объединение (`UNION`) нескольких различных инструкций `SELECT`; допускается использование только одной инструкции `SELECT`.
- Многие запросы, записанные с применением подчиненных запросов, можно также представить в виде многотабличных запросов. Однако имеется также много запросов с подчиненными запросами, которые нельзя выразить в многотабличном виде. К таким относят, например, подчиненные запросы с итоговыми функциями.
- Аналогично главному запросу, в котором может находиться подчиненный запрос, внутри подчиненного запроса может находиться еще один (или более) подчиненный запрос, который называется *вложенным*. С увеличением числа уровней вложенности возрастает время выполнения запроса.

Замечание

В каждой СУБД обязательно имеется оптимизатор, который отвечает за реализацию конкретных запросов пользователя. Как правило, для выполнения отдельного запроса оптимизатор выбирает стратегию на основе, например, следующих положений:

- на какие таблицы есть ссылки в запросе;
- каков размер таблиц;
- какие созданы индексы;
- как выбраны индексы;

- как физически группируются данные на диске;
- какие реляционные операции используются

и т. д.

Примеры

База данных "Студенты"

Пример. Схема базы данных представлена на рис. 4.3.

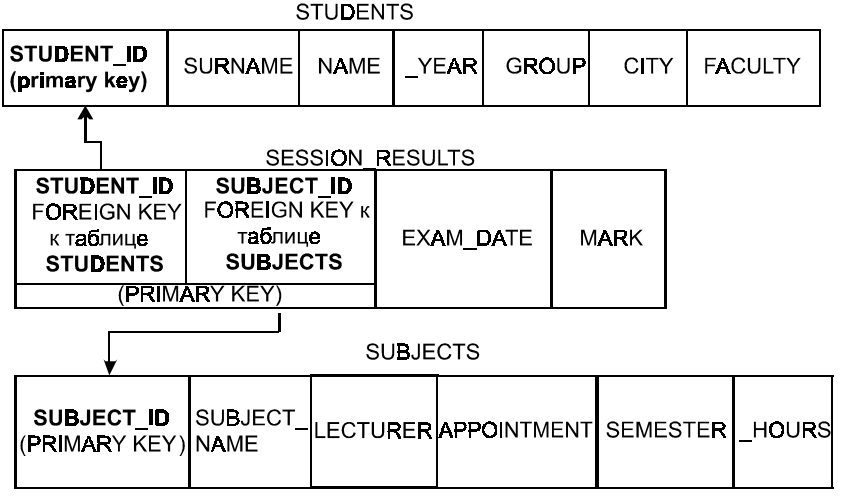


Рис. 4.3. Схема базы данных "Студенты"

Записать на языке SQL следующий запрос к базе.

Вывести результаты экзамена по дисциплине "Экспертные системы" студентов 3 курса математического факультета зимней сессии в виде таблицы со следующими полями: `SURNAME`, `NAME`, `MARK`, `EXAM_DATE`. Провести сортировку по возрастанию поля `SURNAME` (фамилии) (по алфавиту) и по убыванию поля `EXAM_DATE` (дата сдачи).

Решение.

```
SELECT SURNAME, NAME, MARK, EXAM_DATE
  FROM STUDENTS, SUBJECTS, SESSION_RESULTS
  WHERE STUDENTS.STUDENT_ID = SESSION_RESULTS.STUDENT_ID
     AND SUBJECTS.SUBJECT_ID = SESSION_RESULTS.SUBJECT_ID
     AND STUDENTS._YEAR = 3
     AND STUDENTS.FACULTY = 'математический'
     AND SUBJECTS.SUBJECT_NAME = 'экспертные системы'
     AND SUBJECTS.SEMESTER = 5
  ORDER BY 1 ASC, 4 DESC;
```

Пример. Схема базы данных представлена на рис. 4.3. Записать на языке SQL следующий запрос к базе.

Получить список студентов, обучающихся на историческом, экономическом, юридическом или биологическом факультетах, проживающих в городе Гродно и получивших хотя бы по любому предмету оценки 8, 9 или 10. Результаты представить в виде таблицы со следующими полями: `SURNAME`, `NAME`, `SUBJECT_NAME`, `MARK`, `EXAM_DATE`, `FACULTY`. Отсортировать результаты запроса по возрастанию (по алфавиту) поля `FACULTY` и по убыванию поля `MARK`.

Решение.

```
SELECT SURNAME, NAME, SUBJECT_NAME, MARK, EXAM_DATE,
       FACULTY
  FROM SUBJECTS, STUDENTS, SESSION_RESULTS
  WHERE STUDENTS.STUDENT_ID = SESSION_RESULTS.STUDENT_ID
     AND SUBJECTS.SUBJECT_ID = SESSION_RESULTS.SUBJECT_ID
     AND STUDENTS.FACULTY IN ('исторический',
     'экономический', 'юридический', 'биологический')
     AND STUDENTS.CITY = 'Гродно'
     AND SESSION_RESULTS.MARK IN (8, 9, 10)
  ORDER BY 6 ASC, 4 DESC;
```

Пример. Схема базы данных представлена на рис. 4.3. Записать на языке SQL следующий запрос к базе.

Получить средний балл успеваемости по каждому предмету для экономического, математического и биологического факультетов студентов 1, 2, 3 и 4 курсов. Результаты представить в виде таблицы со следующими полями: SUBJECT_NAME, _YEAR, FACULTY, AVG_MARK. Отсортировать полученные результаты по возрастанию (по алфавиту) для полей SUBJECT_NAME и по убыванию для поля AVG_MARK.

Решение.

```
SELECT SUBJECT_NAME, _YEAR, FACULTY, AVG (MARK) AS
       AVG_MARK
  FROM SUBJECTS, STUDENTS, SESSION_RESULTS
  WHERE STUDENTS.STUDENT_ID = SESSION_RESULTS.STUDENT_ID
     AND SUBJECTS.SUBJECT_ID = SESSION_RESULTS.SUBJECT_ID
     AND STUDENTS.FACULTY IN ('математический',
     'экономический', 'биологический')
     AND STUDENTS._YEAR BETWEEN 1 AND 4
  GROUP BY SUBJECT_NAME, _YEAR, FACULTY
  ORDER BY 1 ASC, 4 DESC;
```

Пример *(рекурсивный запрос).* Схема базы данных представлена на рис. 4.4.

Записать на языке SQL следующий запрос к базе.

Получить список студентов экономического факультета 4 и 5 курсов, включающих фамилию старосты. Результаты представить в виде таблицы со следующими полями: STUDENT_SURNAME, NAME, _YEAR, _GROUP, FACULTY, THE_HEAD. Отсортировать полученный список по алфавиту полей STUDENT_SURNAME, NAME и по убыванию поля _GROUP.

Решение.

```
SELECT STUDENT_1.SURNAME AS STUDENT_SURNAME,
       STUDENT_1.NAME, STUDENT_1._YEAR, STUDENT_1._GROUP,
```

```
      STUDENT_1.FACULTY, STUDENT_2.SURNAME AS THE_HEAD
FROM STUDENTS STUDENT_1, STUDENTS STUDENT_2
WHERE STUDENT_1.THE_HEAD_ID = STUDENT_2.STUDENT_ID
                AND STUDENT_1.FACULTY = 'экономический'
                AND STUDENT_1._YEAR IN (4, 5)
ORDER BY 1 ASC, 2 ASC, 4 DESC;
```

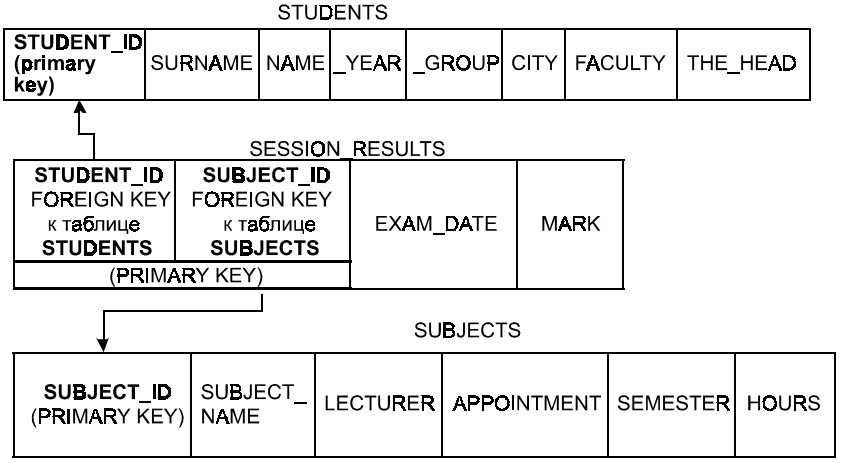


Рис. 4.4. Схема базы данных "Студенты" (обобщенная)

База данных "Продажа товаров"

База данных "Продажа товаров" (рис. 4.5) содержит следующие таблицы.

- `CLIENTS` (КЛИЕНТЫ) — информация о клиентах, которые покупают товары компании (`CLIENT_ID` (номер клиента) — `primary key`, `COMPANY` (юридическое название), `CUSTOMER_REP` (номер_продавца_клиента) — `foreign key` к таблице `EMPLOYEE` (`EMP_ID`), `CREDIT_LIMIT` (лимит_кредита));

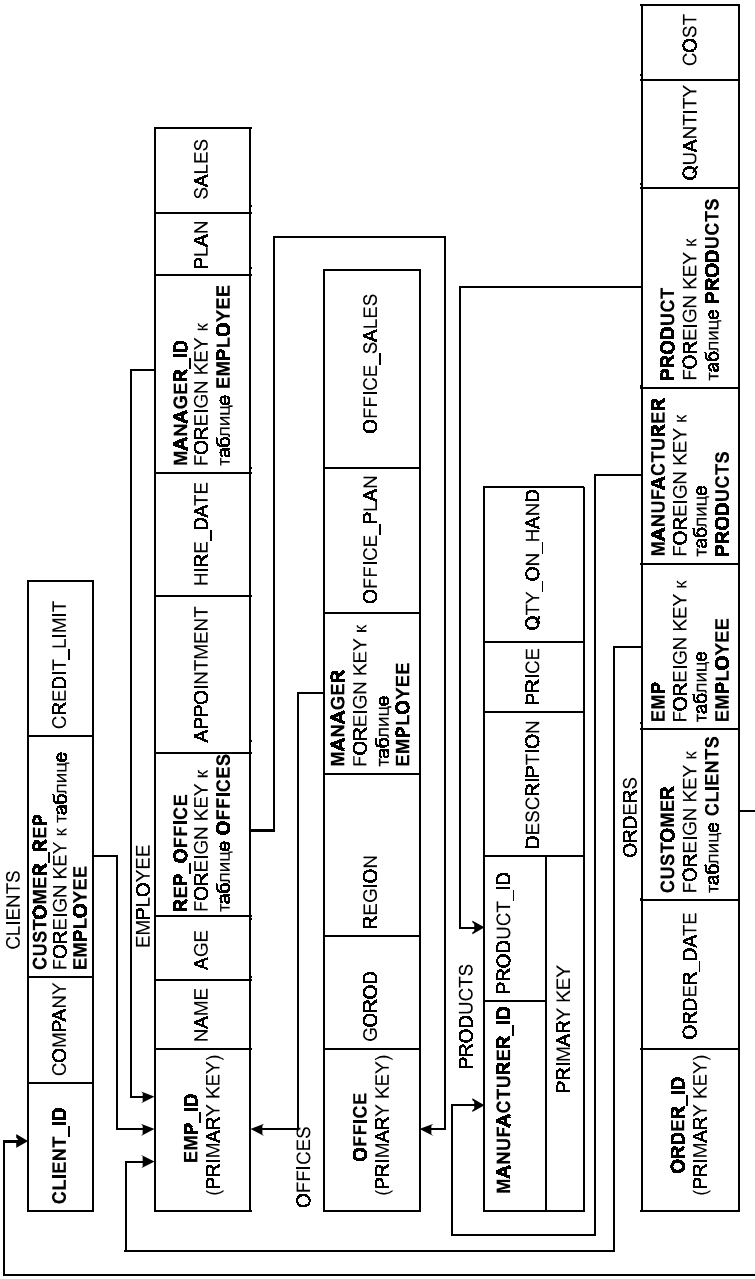


Рис. 4.5. Схема базы данных "Продажа товаров"

- EMPLOYEE (СЛУЖАЩИЕ) — информация о служащих, работающих в компании и продающих товары клиентам (EMP_ID (номер_служащего) — primary_key, NAME (ФИО_служащего), AGE (возраст), REP_OFFICE (номер_офиса) — foreign key к таблице OFFICES (OFFICE) APPOINTMENT (должность), HIRE_DATE (дата_найма), MANAGER_ID (номер_менеджера) — foreign key к таблице EMPLOYEE (EMP_ID), PLAN (план), SALES (продажи));
- OFFICES (ОФИСЫ) — офисы компании, в которых работают служащие (OFFICE (номер_офиса) — primary_key, CITY (город), REGION (регион), MANAGER (номер_менеджера) — foreign key к таблице EMPLOYEE (EMP_ID), OFFICE_PLAN (план_офиса), OFFICE_SALES (продажи_офиса));
- PRODUCTS (ТОВАРЫ) — информация о товарах, продаваемых компанией (MANUFACTURER_ID (номер_производителя) — primary_key, PRODUCT_ID (номер_товара) — primary_key, DESCRIPTION (описание), PRICE (цена), QTY_ON_HAND (количество_на_складе));
- ORDERS (ЗАКАЗЫ) — информация о заказах, сделанных клиентом (ORDER_ID (номер_заказа) — primary_key, ORDER_DATE (дата_заказа), CUSTOMER (клиент) — foreign key к таблице CLIENTS (CLIENT_ID), EMP (служащий) — foreign key к таблице EMPLOYEE (EMP_ID), MANUFACTURER (производитель) — foreign key к таблице PRODUCTS (MANUFACTURER_ID), PRODUCT (товар) — foreign key к таблице PRODUCTS (PRODUCT_ID), QUANTITY (количество), COST (стоимость)). Для простоты предполагается, что один заказ может содержать один товар.

Пример *(объединение результатов нескольких запросов).* Вывести список всех товаров, цена которых превышает 4000 руб. или которых было заказано более чем на 50 000 руб. за один раз.

Решение.

```
SELECT MANUFACTURER_ID, PRODUCT_ID
  FROM PRODUCT
```

```
  WHERE PRICE > 4000.00
  UNION ALL
SELECT DISTINCT MANUFACTURER, PRODUCT
  FROM ORDERS
  WHERE COST > 50000.00;
```

По умолчанию операция UNION в процессе своего выполнения удаляет повторяющиеся строки. Если необходимо сохранить повторяющиеся строки, после UNION ставят предикат ALL. Обработка повторяющихся строк в операции UNION и инструкции SELECT осуществляется по-разному. SELECT по умолчанию оставляет такие строки (SELECT ALL). Чтобы их удалить, необходимо явно указать предикат DISTINCT.

Пример *(объединение результатов нескольких запросов).* Вывести список всех товаров, цена которых превышает 4000 руб. или которых было заказано более чем на 50 000 руб. за один раз; список отсортировать по наименованию производителя и номеру товара.

Решение.

```
SELECT MANUFACTURER_ID, PRODUCT_ID
  FROM PRODUCT
  WHERE PRICE > 4000.00
  UNION
SELECT DISTINCT MANUFACTURER, PRODUCT
  FROM ORDERS
  WHERE COST > 50000.00
  ORDER BY 1, 2;
```

Пример *(рекурсивное объединение).* Вывести список служащих, планы которых превышают планы их руководителей.

Решение.

```
SELECT EMPLOYEE.NAME, EMPLOYEE.PLAN, MANAGERS.PLAN
  FROM EMPLOYEE, EMPLOYEE MANAGERS
```

```
  WHERE EMPLOYEE.MANAGER_ID = MANAGERS.EMP_ID
     AND EMPLOYEE.PLAN > MANAGERS.PLAN;
```

Пример *(рекурсивное объединение).* Вывести список служащих, которые работают со своими руководителями в разных офисах, включая имена и офисы как служащих, так и руководителей.

Решение.

```
SELECT EMPS.NAME, EMP_OFFICE.CITY, MANAGERS.NAME,
       MANAGER_OFFICE.CITY
  FROM EMPLOYEE EMPS, EMPLOYEE MANAGERS,
       OFFICES EMP_OFFICE, OFFICES MANAGER_OFFICE
  WHERE EMPS.REP_OFFICE = EMP_OFFICE.OFFICE
     AND MANAGERS.REP_OFFICE = MANAGER_OFFICE.OFFICE
     AND EMPS.MANAGER_ID = MANAGERS.EMP_ID
     AND EMPS.REP_OFFICE < > MANAGERS.REP_OFFICE;
```

Пример *(левое внешнее объединение таблиц).* Вывести список служащих и городов, где они работают. Организовать запрос таким образом, чтобы были видны также и служащие, которые приняты на работу, но еще официально не назначены в офис конкретного города.

Решение.

```
SELECT NAME, CITY
  FROM EMPLOYEE, OFFICES
  WHERE REP_OFFICE *= OFFICE;
```

Таблица, у которой все строки включаются в объединение, называется *главной*, а таблица, в которой имеются недостающие элементы — значения `NULL`, *вспомогательной*. В данном случае таблица `EMPLOYEE` является главной, таблица `OFFICES` — вспомогательной. Значения `NULL` могут содержаться в столбце `CITY`.

Пример *(правое внешнее объединение таблиц).* Вывести список офисов и служащих, работающих в каждом из них.

Решение.

```
SELECT NAME, CITY
   FROM EMPLOYEE, OFFICES
   WHERE OFFICE =* REP_OFFICE;
```

В данном случае таблица `OFFICES` является главной, а таблица `EMPLOYEE` — вспомогательная. В столбце `NAME` могут присутствовать `NULL`-значения, т. е. имеется город, в котором на данный момент нет служащих.

Пример *(условия отбора групп).* Получить цену, количество на складе и общее количество заказанных единиц для каждого наименования товара, если для него общее количество заказанных единиц превышает 68% от количества товара на складе.

Решение.

```
SELECT DESCRIPTION, PRICE, QTY_ON_HAND, SUM(QUANTITY)
  FROM PRODUCTS, ORDERS
  WHERE MANUFACTURER = MANUFACTURER_ID
      AND PRODUCT = PRODUCT_ID
  GROUP BY MANUFACTURER_ID, PRODUCT_ID, DESCRIPTION,
           PRICE, QTY_ON_HAND
  HAVING SUM (QUANTITY) > (0.68 * QTY_ON_HAND)
  ORDER BY QTY_ON_HAND DESC;
```

Пример *(подчиненные запросы в предложении `WHERE`).* Получить список имеющихся в наличии товаров от компании ARGO, количество которых превышает количество товара software4087.

Решение.

```
SELECT DESCRIPTION, QTY_ON_HAND
   FROM PRODUCTS
```

```
   WHERE MANUFACTURER_ID = 'Microsoft' AND QTY_ON_HAND >
(SELECT QTY_ON_HAND
        FROM PRODUCT
        WHERE MANUFACTURER_ID = 'Intel'
        AND PRODUCT_ID = 'software4087');
```

Следует заметить, что подчиненный запрос в операции сравнения может стоять только справа от оператора сравнения, т. е.

выражение { = | <> | < | <= | > | >= } (подчиненный запрос)

Подчиненный запрос слева от операции сравнения запрещен, т. е.

(подчиненный запрос) { = | <> | < | <= | > | >= } выражение

Пример *(подчиненный запрос в предложении* `WHERE`*; проверка на принадлежность результатов подчиненного запроса; предикат* `IN`*).* Получить список всех клиентов, заказавших изделия компании BETA (идентификатор изготовителя — `BETA`, идентификаторы товаров начинаются с номера 39) в период между январем и июнем 2005 года.

Решение.

```
SELECT COMPANY
   FROM CLIENTS
   WHERE CLIENT_ID IN (SELECT DISTINCT CUSTOMER
                       FROM ORDERS
                       WHERE MANUFACTURER = 'BETA'
                       AND PRODUCT LIKE '39%'
                       AND ORDER_DATA BETWEEN
                       '01-JAN-05' AND '30-JUN-05');
```

Пример *(подчиненный запрос в предложении* `WHERE`*; проверка на существование; предикат* `EXISTS`*).* Получить список товаров, на которые получен заказ стоимостью 120 000 руб. или больше.

Решение.

Запрос можно сформулировать иначе: получить список товаров, для которых в таблице `ORDERS` существует, по крайней мере, один заказ, удовлетворяющий условиям:

- является заказом на данный товар;
- имеет стоимость не менее 120 000 руб.

```
SELECT DESCRIPTION
   FROM PRODUCTS
   WHERE EXISTS (SELECT *
                 FROM ORDERS
                 WHERE PRODUCT = PRODUCT_ID
                 AND MANUFACTURER = MANUFACTURER_ID
                 AND COST > = 120000.00);
```

В результате проверки на существование можно выяснить, содержится ли в таблице результатов подчиненного запроса хотя бы одна строка. Аналогичной простой проверки не существует. Проверка на существование допустима только в подчиненных запросах.

Пример *(подчиненный запрос в предложении* `WHERE`*; проверка на существование; предикат* `EXISTS`*).* Получить список клиентов, обслуживаемых Ивановым, которые не сделали заказы на сумму свыше 5000 руб.

Решение.

```
SELECT COMPANY
  FROM CLIENTS
  WHERE CUSTOMER_REP = (SELECT EMP_ID
                        FROM EMPLOYEE
                        WHERE NAME = 'Ivanov'
                          AND NOT EXISTS (SELECT *
                                FROM ORDERS
                                WHERE CUSTOMER = CLIENT_ID
                                    AND COST > 5000.00);
```

Пример *(подчиненный запрос в предложении* `WHERE`*; проверка на существование; предикат* `EXISTS`*).* Вывести список городов, где имеется служащий, чей план превышает 60% от плана офиса.

Решение.

```
SELECT CITY
   FROM OFFICES
   WHERE EXISTS (SELECT *
                 FROM EMPLOYEE
                 WHERE REP_OFFICE = OFFICE
                    AND PLAN > (0.6 * OFFICE_PLAN));
```

Пример *(подчиненный запрос в предложении* `WHERE`*; многократное сравнение; предикат* `ANY`*).* Получить список служащих, которые оформили заказ на сумму большую, чем 15% от их плана.

Решение.

```
SELECT NAME
   FROM EMPLOYEE
   WHERE 0.15 * PLAN < ANY (SELECT COST
                            FROM ORDERS
                            WHERE EMP = EMP_ID);
```

В проверке IN выясняется, не равно ли некоторое значение одному из значений, содержащихся в столбце результатов подчиненного запроса. В SQL существуют разновидности многократного сравнения — ANY и ALL, расширяющие предыдущую проверку до уровня других операторов сравнения, таких как больше (>) или меньше (<). В обеих проверках некоторое значение сравнивается со столбцом данных, отобранных подчиненным запросом.

Предикат ANY. В проверке ANY, чтобы сравнить проверяемое значение со столбцом данных, отобранных подчиненным запросом, используется один из шести операторов сравнения (=, < >, <, < =, >, > =). Проверяемое значение поочередно сравнивается с каждым элементом, содержащимся в столбце. Если любое из этих сравнений дает результат TRUE, то проверка ANY возвращает значение TRUE.

Пример *(подчиненный запрос в предложении WHERE; многократное сравнение; предикат ALL).* Получить список тех офисов и их плановые объемы продаж для всех служащих, у которых фактический объем продаж превышает 55% от плана офиса.

Решение.

```
SELECT CITY, OFFICE_PLAN
   FROM OFFICES
   WHERE (.55* OFFICE_PLAN) < ALL (SELECT SALES
                                   FROM EMPLOYEE
                                   WHERE REP_OFFICE = OFFICE);
```

Предикат ALL. В проверке ALL используется один из операторов сравнения (=, <>, <, <=, >, >=) для сравнения проверяе-

мого значения со столбцом данных, отобранных подчиненным запросом. Проверяемое значение поочередно сравнивается с каждым элементом, содержащимся в столбце. Если все сравнения дают результат TRUE, то проверка ALL возвращает значение TRUE.

Задания

1. Какие типы команд можно выделить в языке SQL?
2. Что такое инструкция SQL?
3. Чем инструкция SQL отличается от команды и предложения?
4. Когда используются полные имена таблиц?
5. Охарактеризовать наиболее употребительные типы данных SQL.
6. Привести пример создания пользовательского типа данных.
7. Для чего используются домены? Привести пример создания домена.
8. Константы какого типа используются в SQL?
9. Что такое выражение?
10. Охарактеризовать основные типы функций SQL.
11. Что включает схема в SQL?
12. Записать инструкцию на SQL для создания таблицы СЕКЦИЯ со следующими полями: код секции, название секции, цена занятия, количество занятий в неделю, название зала, фамилия тренера. Поле код секции является первичным ключом таблицы СЕКЦИЯ. Значения поля количество занятий в неделю находятся в пределах от 1 до 7, по умолчанию этому полю присваивается значение 2. Значение поля секция берется из списка {аэробика, атлетика, теннис, бокс, дзюдо, плавание, шейпинг}. Поля код секции, название секции, название зала не могут принимать значений NULL.

13. Записать инструкцию на SQL для создания таблицы УЧАСТНИКИ со следующими полями: код участника, фамилия, имя, отчество, адрес, учебное заведение, спортивный разряд. Поле код участника является первичным ключом таблицы УЧАСТНИКИ. Значение поля учебное заведение выбирается из списка (сформировать произвольный список учебных заведений). Значения поля спортивный разряд находятся в пределах от 1 до 6, по умолчанию этому полю присваивается значение 6. Поля код участника, фамилия, имя, отчество не могут принимать значений NULL.

14. Записать инструкцию на SQL для создания таблицы ПОСЕЩЕНИЕ_СЕКЦИИ со следующими полями: код секции, код участника, № квитанции, дата оплаты, количество посещений. Поле № квитанции является первичным ключом таблицы ПОСЕЩЕНИЕ_СЕКЦИИ. Поля код секции, код участника не могут принимать значений NULL. Поле код секции является внешним ключом к таблице СЕКЦИЯ, а поле код участника — внешним ключом к таблице УЧАСТНИКИ (названия связей определить произвольно). Для полей код секции и код участника установить режимы обеспечения целостности, запрещающие удаление и обновление соответствующих родительских сущностей.

15. Записать инструкцию на SQL для создания таблицы УЧАСТНИКИ_1 со следующими полями: код участника, фамилия, имя, отчество, адрес, учебное заведение, спортивный разряд, фамилия тренера. Значение поля фамилия тренера должно быть идентификатором участника, являющегося тренером группы, в которой занимается конкретный участник, для чего следует указать необходимые ограничения. Поле код участника является первичным ключом таблицы УЧАСТНИКИ_1. Значение поля учебное заведение выбирается из списка (сформировать произвольный список учебных заведений). Значения поля спортивный разряд находятся в пределах от 1 до 6, по умолчанию этому полю присваивается значение 6. Поля код участника, фамилия, имя, отчество не могут принимать значений NULL.

16. База данных представлена на рис. 4.3. Записать на языке SQL следующие запросы к базе.

- Вывести результаты экзамена по дисциплинам "Экспертные системы" и "Системное программное обеспечение" студентов 3 курса математического факультета, получивших оценки 6, 7 или 8, в виде таблицы со следующими полями: SURNAME, NAME, MARK, EXAM_DATE. Провести сортировку по возрастанию поля SURNAME (фамилии) (по алфавиту) и по убыванию поля EXAM_DATE (дата сдачи).

- Узнать, имеются ли предметы, читаемые преподавателями Ивановым или Петровым в 5 семестре для студентов математического факультета, количество часов отведенное на которые находится в пределах от 30 до 70. Результаты представить в виде таблицы со следующими полями: SUBJECT_NAME, LECTURER, _HOURS. Отсортировать результаты запроса по возрастанию (по алфавиту) поля SUBJECTS и по убыванию поля _HOURS.

- Узнать, имеются ли предметы, читаемые преподавателями Котовым, Сидоровым или Петровым в 3 и 4 семестрах для студентов математического факультета, количество часов отведенное на которые находится в пределах меньше 50. Результаты представить в виде таблицы со следующими полями: SUBJECT_NAME, LECTURER, _HOURS. Отсортировать результаты запроса по возрастанию (по алфавиту) поля SUBJECTS и по убыванию поля _HOURS.

- Получить список студентов всех курсов математического факультета, получивших на экзаменах неудовлетворительные оценки (<= 3 по 10-балльной системе), которые сдавались в летнюю сессию. Результаты представить в виде таблицы со следующими полями: SURNAME, NAME, SUBJECT_NAME, LECTURER, MARK, EXAM_DATE. Отсортировать результаты запроса по убыванию поля MARK.

- Получить список студентов вторых курсов математического и экономического факультетов, получивших 9-ти и 10-балльные оценки на экзаменах, которые сдавались в летнюю сессию. Результаты представить в виде таблицы со следующими полями: SURNAME, NAME, SUBJECT_NAME, LECTURER, MARK, EXAM_DATE. Отсортировать результаты запроса по убыванию поля MARK.

- Получить список студентов, обучающихся на историческом, экономическом или юридическом факультетах, проживающих в Минске или Москве и получивших хотя бы по любому предмету оценки 8 или 9. Результаты представить в виде таблицы со следующими полями: SURNAME, NAME, SUBJECT_NAME, MARK, EXAM_DATE, FACULTY. Отсортировать результаты запроса по возрастанию (по алфавиту) поля FACULTY и по убыванию поля MARK.

- Получить средний балл успеваемости для каждого студента математического факультета 2, 3 и 4 курсов. Результаты представить в виде таблицы со следующими полями: SURNAME, NAME, AVERAGE_MARK, _YEAR. Отсортировать полученные результаты по возрастанию (по алфавиту) для полей SURNAME, NAME и по убыванию для поля AVERAGE_MARK.

- Получить средний балл успеваемости для каждого студента экономического, биологического или исторического факультетов 3 и 4 курсов. Результаты представить в виде таблицы со следующими полями: SURNAME, NAME, AVERAGE_MARK, _YEAR. Отсортировать полученные результаты по возрастанию (по алфавиту) для полей SURNAME, NAME и по убыванию для поля AVERAGE_MARK.

- Получить список студентов экономического факультета 4 и 5 курсов, включающих фамилию старосты. Результаты представить в виде таблицы со следующими полями: STUDENT_SURNAME, NAME, _YEAR, _GROUP, FACULTY, THE_HEAD. Отсортировать полученный список по алфавиту полей STUDENT_SURNAME, NAME и по убыванию поля _GROUP.

- Получить список студентов математического факультета 3, 4 и 5 курсов, включающих фамилию старосты. Результаты представить в виде таблицы со следующими полями: STUDENT_SURNAME, NAME, _YEAR, _GROUP, FACULTY, THE_HEAD. Отсортировать полученный список по алфавиту полей STUDENT_SURNAME, NAME и по убыванию поля _GROUP.

- Получить список предметов, которые сдавались в зимнюю сессию студентами математического факультета 4 курса, количество часов на изучение которых превышает 60. Результаты представить в виде таблицы со следующими полями: SUBJECT_NAME, LECTURER, _HOURS. Отсортировать полученные результаты по возрастанию (по алфавиту) для полей SUBJECT_NAME и по убыванию для поля _HOURS.

- Получить список предметов, которые сдавались в летнюю сессию студентами математического факультета 4 курса, количество часов на изучение которых более 60, но менее 100. Результаты представить в виде таблицы со следующими полями: SUBJECT_NAME, LECTURER, _HOURS. Отсортировать полученные результаты по возрастанию (по алфавиту) для полей SUBJECT_NAME и по убыванию для поля _HOURS.

Глава 5

Вспомогательные аспекты баз данных

При рассмотрении различных систем управления базами данных большое значение имеют такие аспекты, как хранение и защита данных от разрушений, возможность обновления данных, сохранение точности и корректности хранимых данных, многопользовательский доступ к данным и т. п. Учет перечисленных и некоторых других сторон при организации функционирования системы управления базами данных позволяет создавать достаточно надежные и защищенные системы, поддерживающие хранение и обработку данных в течение длительного времени.

Целостность баз данных

Целостность (integrity — точность, истинность) *данных* предполагает корректное выполнение действий, разрешенных пользователям. Обычно ограничения целостности применяются для описания *базовых* отношений. Производные отношения автоматически наследуют некоторые ограничения целостности отношений, из которых они выведены. Однако для произвольных отношений, кроме автоматически наследуемых

ограничений могут быть заданы также некоторые дополнительные ограничения целостности.

Если база данных находится в состоянии целостности, то это означает ее *корректность*, т. е. отсутствие нарушений всех известных ограничений. Аспекты целостности необходимо учитывать как при проектировании базы данных, так и во время ее использования. Ограничения целостности, как и правила безопасности, хранятся в системном каталоге. Они приводятся в действие подсистемой целостности СУБД, которая отвечает за отслеживание выполняемых пользователем операций (`INSERT`, `UPDATE`, `DELETE`) и гарантирует выполнение только тех, которые не нарушают известных системе ограничений.

Ограничения целостности в общем виде содержат *три компонента*:

```
CREATE INTEGRITY RULE имя
                        ограничение
                      [ ON ATTEMPTED VIOLATION действие ];
```

Параметр `имя` является именем данного ограничения, которое регистрируется в системном каталоге. Ограничения целостности всегда проверяются немедленно при каждой операции обновления.

`Ограничение` — выражение, которому должно удовлетворять ограничение целостности. Ограничение целостности удовлетворяется тогда и только тогда, когда это выражение *истинно*, а нарушается, если это выражение *ложно*.

Директива `ON ATTEMPTED VIOLATION` — *реакция на нарушение*, указывает системе те действия, которые необходимо выполнить при попытке пользователя нарушить ограничение целостности.

Ограничения целостности можно классифицировать по следующим категориям:

- ограничения домена — задаются допустимые значения для данного домена;

- ограничения атрибута — задаются допустимые значения для данного атрибута;
- ограничения таблицы — задаются допустимые значения для данного отношения; как правило, ограничения таблицы ограничивают значения, которые могут быть в записях таблицы;
- целостность по ссылкам — предполагает обеспечение целостности отношений потомок/предок, которые поддерживаются первичными и внешними ключами;
- ограничения базы данных (деловые правила или утверждения) — задают допустимые значения для всей базы данных и связывают в различных выражениях значения столбцов из разных таблиц базы данных.

В стандарте SQL ограничения целостности обычно делятся на три категории:

- ограничения домена;
- ограничения базовой таблицы;
- общие ограничения — "утверждения".

Общие ограничения в SQL создаются с помощью утверждения `CREAT ASSERTION` (*см. главу 4*).

Пример. Поставщикам со статусом меньше 5 нельзя поставлять товары в количестве более 250.

Решение.

```
CREATE ASSERTION Allowed
CHECK (NOT EXISTS (SELECT *
  FROM S, SP
  WHERE S.STATUS < 5
  AND S.S# = SP.S#
  AND SP.QTY > 250));
```

Триггеры

Триггер представляет собой автоматически выполняемый процедурный SQL-код, который активизируется при каком-либо событии, связанном с изменением данных. Триггеры являются важными объектами при работе с базой данных и могут использоваться для следующих целей:

- поддержка ссылочной целостности между таблицами базы данных;
- автоматизация некоторых операций работы с данными и выдача соответствующих корректирующих рекомендаций пользователям базы данных;
- обновление таблиц;
- добавление записей в таблицы;
- вызов хранимых процедур.

Триггеры расширяют возможности системы с базой данных и, как правило, для них справедливо следующее:

- инициализация триггера происходит до или после выбора, вставки или удаления строки данных;
- связь с таблицей базы данных, причем с каждой таблицей базы данных может быть связан один или более триггеров;
- триггер выполняется как часть транзакции, которая его инициировала.

Синтаксис триггера (например, для сервера InterBase) представляется в общем виде следующим образом:

```
CREATE TRIGGER Имя_триггера FOR Имя_таблицы
[ACTIVE | INACTIVE] (BEFORE | AFTER) (UPDATE | INSERT |
DELETE)
    [POSITION Число]
    AS
    BEGIN
        Тело_триггера;
    END
```

В данном случае ключевые слова ACTIVE и INACTIVE определяют активность триггера сразу после его создания. По умолчанию устанавливается параметр ACTIVE, и созданный триггер при наступлении соответствующего события выполняется. Если установлен параметр INACTIVE, триггер является неактивным, и при наступлении события выполняться он не будет. Созданный триггер всегда можно активизировать либо деактивизировать.

Параметры BEFORE и AFTER задают момент начала выполнения триггера — до или после наступления соответствующего события, связанного с изменением записей.

Параметры UPDATE, INSERT и DELETE определяют, при наступлении какого события вызывается триггер: при редактировании, добавлении либо удалении записей соответственно.

Для одного события можно создать несколько триггеров, каждый из которых будет автоматически выполнен (если находится в активном состоянии). При наличии нескольких триггеров порядок их вызова (выполнения) определяет число, указанное в операнде POSITION. Триггеры выполняются в порядке возрастания этих чисел.

Созданный триггер можно удалить либо изменить. Для удаления триггера используется оператор DROP:

```
DROP TRIGGER Имя_триггера;
```

Для изменения применяется оператор ALTER:

```
ALTER TRIGGER Имя_триггера FOR Имя_таблицы
[ACIVE | INACTIVE] (BEFORE | AFTER) (UPDATE | INSERT |
DELETE)
    [POSITION Число]
    AS
    BEGIN
        Тело_триггера;
    END
```

После выполнения оператора ALTER TRIGGER предыдущее описание триггера с заданным именем заменяется новым.

Тело триггера программируется аналогично программированию тела хранимой процедуры, для чего используется специальный язык, позволяющий также создавать хранимые процедуры. Для доступа к значениям столбца используются следующие конструкции:

- `OLD.Имя_столбца` — обращение к старым (до внесения изменений) значениям столбца;
- `NEW.Имя_столбца` — обращение к новым (после внесения изменений) значениям столбца.

Пример. Создать триггер, реализующий каскадное изменение в таблице `SESSION_RESULTS` (Сдача предметов) при изменении названия предметов в таблице `SUBJECTS` (Предметы).

Решение.

```
CREATE TRIGGER REFRESH FOR SUBJECTS
          ACTIVE
             BEFORE UPDATE
             AS
             BEGIN
                IF (OLD.SUBJECT_ID <> NEW.SUBJECT_ID)
                  THEN UPDATE SESSION_RESULTS
                  SET SUBJECT_ID = NEW.SUBJECT_ID
                  WHERE SUBJECT_ID = OLD.SUBJECT_ID;
             END
```

Пример. Создать триггер, реализующий каскадное удаление в таблице `SESSION_RESULTS` при удалении записей в таблице `STUDENTS` (см. рис. 4.3).

Решение.

```
CREATE TRIGGER DELETION FOR STUDENTS
    ACTIVE
    AFTER DELETE
```

```
AS
  BEGIN
    DELETE FROM SESSION_RESULTS
    WHERE SESSION_RESULTS.STUDENT_ID =
     STUDENTS.STUDENT_ID;
  END
```

Создание генераторов

Не все СУБД обеспечивают создание для таблиц автоинкрементного типа данных для обеспечения установки уникальности значений столбца. Поэтому для обеспечения уникальности значений ключевых столбцов используются генераторы.

Генератор создается следующим оператором (сервер InterBase):

```
CREATE GENERATOR Имя_генератора;
SET GENERATOR Имя_генератора TO Начальное_значение;
```

Начальное значение представляет собой целое число, начиная с которого формируются уникальные значения генератора.

Обращение к генератору происходит с помощью функции:

```
GEN_ID (Имя_генератора, Шаг);
```

Данная функция возвращает значение, увеличенное на целочисленный шаг относительно предыдущего значения генератора.

Замечание

После определения начального значения генератора изменять его нельзя. Также нельзя изменять шаг. Невыполнение этого правила приведет к тому, что генератор может возвратить неуникальное значение со всеми вытекающими последствиями.

Пример. Создать генератор, обеспечивающий уникальные значения для столбца `SUBJECT_ID` таблицы `SUBJECTS` (см. рис. 4.3).

Решение.

1. Создается генератор `SUBJ_GNR` с начальным значением, равным десяти:

```
CREATE GENERATOR SUBJ_GNR;
SET GENERATOR SUBJ_GNR TO 10;
```

2. Обращения к созданному генератору можно реализовать в теле триггера:

```
CREATE TRIGGER STUD_SUBJ FOR SUBJECTS
    ACTIVE
    BEFORE INSERT
    AS
    BEGIN
        NEW.SUBJECT_ID = GEN_ID(SUBJ_GNR, 2);
    END
```

Хранимые процедуры

Хранимая процедура представляет собой подпрограмму, которая располагается, как правило, на сервере и вызывается из приложения клиента. Использование хранимых процедур увеличивает скорость доступа к базе данных. Преимущества использования хранимых процедур:

- при передаче по сети серверу передается достаточно короткое имя обращения к хранимой процедуре вместо текста запроса, который намного длиннее данного имени;
- хранимая процедура не требует предварительной синтаксической проверки (в отличие от запроса);
- хранимые процедуры являются общими для всех приложений-клиентов и реализуют единые правила работы с базой данных;
- для выполнения хранимой на сервере процедуры в приложении используется компонент `StoredProc`.

Синтаксис хранимой процедуры (сервер InterBase) может быть представлен в следующем виде (включая использование соответствующих разделителей — "," и ";"):

```
CREATE PROCEDURE Имя_процедуры
/* Список входных параметров: */
                (<Имя_параметра> <Тип_параметра>,
                  ...,
                <Имя_параметра> <Тип_параметра>)
        RETURNS
/* Список выходных параметров: */
                 (<Имя_параметра> <Тип_параметра>,
                  ...,
                <Имя_параметра> <Тип_параметра>)
         AS
/* Начало Тела процедуры */
/* Объявление переменных (описательная часть Тела
процедуры) */
                    DECLARE VARIABLE <Имя_переменной_1>
<Тип_переменной_1>;
                      ...;
                    DECLARE VARIABLE <Имя_переменной_n>
<Тип_переменной_n>;
           BEGIN
/* Исполнительная часть Тела процедуры */
              <Оператор_1>;
                ...;
              <Оператор_n>
           END
/* Конец Тела процедуры */
```

После имени процедуры идет необязательный список входных параметров, с помощью которых из приложения в процедуру передаются исходные данные. После ключевого слова `RETURNS` указывается список выходных параметров, с помо-

щью которых в приложение передаются результаты выполнения процедуры.

При задании имен процедур и параметров рекомендуется использовать мнемонические правила. Так, например, имя процедуры можно начинать с префикса `p` (от Procedure), имя входного параметра — с префикса `ip` (от Input Parameter), имя выходного параметра — с префикса `op` (от Output Parameter) и т. д. При использовании параметра в выражениях тела процедуры перед его именем следует указывать знак ":".

Тело процедуры состоит из двух частей: описательной и исполнительной. В описательной части объявляются переменные, которые используются внутри процедуры. Данные переменные являются локальными и используются только внутри конкретной процедуры. В исполнительной части находится совокупность операторов (либо, как минимум — один), выполняющих необходимые действия. По аналогии с языком Pascal, операторы заключаются в операторные скобки `BEGIN` и `END`.

Созданную процедуру можно удалить или изменить. Для удаления процедуры используется оператор `DROP`:

```
DROP PROCEDURE Имя_процедуры;
```

Изменение процедуры выполняется с помощью оператора `ALTER`:

```
ALTER PROCEDURE Имя_процедуры
/* Список входных параметров: */
                (<Имя_параметра> <Тип_параметра>,
                  ...,
                <Имя_параметра> <Тип_параметра>)
        RETURNS
/* Список выходных параметров: */
                 (<Имя_параметра> <Тип_параметра>,
                  ...,
                <Имя_параметра> <Тип_параметра>)
         AS
```

```
/* Начало Тела процедуры */
/* Объявление переменных (описательная часть Тела
процедуры) */
                    DECLARE VARIABLE <Имя_переменной_1>
<Тип_переменной_1>;
                      ...;
                    DECLARE VARIABLE <Имя_переменной_n>
<Тип_переменной_n>;
           BEGIN
/* Исполнительная часть Тела процедуры */
              <Оператор_1>;
                ...;
              <Оператор_n>
           END
/* Конец Тела процедуры */
```

После выполнения оператора `ALTER PROCEDURE` предыдущее описание хранимой процедуры с указанным именем заменяется на новое.

Для написания хранимых процедур и триггеров используется язык хранимых процедур, представляющий процедурный алгоритмический язык. Синтаксис языка хранимых процедур сервера InterBase похож на синтаксис языка Pascal (лежащего также в основе Delphi).

Проведем краткий обзор составляющих языка хранимых процедур.

В текст процедуры (триггера) можно добавлять комментарии для удобства восприятия листинга процедуры (триггера). Для определения комментариев используются символы слэш, наклоненный вправо, и звездочка: "`/*`" и "`*/`".

Операторы, которые используются в теле процедуры, представлены в табл. 5.1. Каждый оператор должен заканчиваться точкой с запятой (кроме составного оператора).

Таблица 5.1. *Операторы языка хранимых процедур*

Оператор	Описание	Пример

Оператор объявления переменных
`DECLARE VARIABLE <Имя_переменной> <Тип_переменной>;`
	Допустимыми типами для переменных являются типы столбцов InterBase, например, `varchar` либо `date`.
	Переменные являются локальными и действуют только внутри соответствующей процедуры
		`DECLARE VARIABLE Number INTEGER;`
		`DECLARE VARIABLE Str VARCHAR(20);`
		`DECLARE VARIABLE WDate DATE;`
		В данном случае объявляются три переменных `Number`, `Str` и `WDate` следующих типов: целый, строка и дата соответственно

Оператор присваивания
`<Имя переменной> = <Выражение>;`
	При выполнении оператора присваивания вычисляется значение выражения и присваивается переменной, имя которой задано в левой части. Можно указывать имена локальных переменных и выходных параметров, перед именем выходного параметра двоеточие не ставится. Переменная и выражение обязаны иметь совместимые типы (иначе при выполнении процедуры возникает ошибка). В качестве операндов выражения можно использовать значения, имена переменных и параметров, встроенные и пользовательские функции, а также вызовы генераторов	`DECLARE VARIABLE d INTEGER;`
		`DECLARE VARIABLE x VARCHAR(20);`
		`d = 48;`
		`x = UPPER(s) ;`
		Целочисленной переменной `d` присваивается значение 48, а символы строки `x` переводятся в верхний регистр

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример

Составной оператор
`BEGIN <Оператор_1>;`
`...;`
`<Оператор_N>`
`END`
	Представляет собой группу из произвольного числа любых операторов, ограниченную операторными скобками `begin` и `end`.
	
	Составной оператор может располагаться в любом месте, где допускается наличие оператора. Наиболее часто составной оператор используется в условных операторах и операторах цикла. В конце составного оператора (перед `END`) точка с запятой не ставится. Составной оператор объединяет также все операторы тела процедуры или триггера
		Пример использования условного и составного операторов:
		
		`IF (x < 5)`
		`  THEN BEGIN d = 0;`
		`  t = ""`
		`  END`
		`  ELSE d = 55;`
		
		Если значение числовой переменной `x` меньше пяти, то значения переменных `d` и `t` сбрасываются (соответствующие операторы присваивания объединены в составной оператор). Если условие не выполняется, то числовой переменной `d` присваивается значение 55, а значение строковой переменной `t` остается без изменений

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример
Условный оператор `IF (<Условие>) THEN <Оператор_1> [ELSE <Оператор_2>] ;`	Позволяет организовать ветвление при вычислениях. Условие представляет собой выражение логического типа. Условный оператор выполняется следующим образом: если условие истинно (т. е. имеет значение `TRUE`), то выполняется `Оператор_1`, в противном случае выполняется `Оператор_2`. Данные операторы могут быть составными. Возможно применение условного оператора в сокращенной форме: слово `ELSE` и оператор, следующий за ним, отсутствуют. В таком случае при невыполнении условия управление сразу же передается на оператор, следующий за условным	

Оператор цикла (оператор повтора)
`WHILE (<Условие>) DO <Оператор>;`
	Обеспечивает повторение группы операторов при соблюдении определенных условий.
	
	Оператор, расположенный после слова `DO`, будет выполняться до тех пор, пока соблюдается указанное условие. Если в цикле требуется выполнить несколько операторов, то их нужно заключить в операторные скобки
		`S = 0;`
		`n = 1;`
		`WHILE (n <= 10) DO`
		`  BEGIN`
		`  s = s + n;`
		`  n = n + 1`
		`  END`

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример
		Выполняется подсчет суммы чисел от 1 до 10. Перебор чисел и накапливание суммы осуществляются в цикле

Оператор выбора записи (строки)
`INTO :<Имя>, :..., :<Имя>;`
	Представляет собой оператор `SELECT`, который возвращает одну строку. Значения столбцов возвращаемой строки присваиваются указанным переменным или параметрам. При своем выполнении оператор выбора записи (аналогично оператору `SELECT`) отбирает несколько строк, но возвращает только одну строку.
	Имя, которое следует после двоеточия, указывает переменную или выходной параметр, которым должны быть присвоены значения столбцов строки, полученной в результате выполнения оператора `SELECT`.
	Данный оператор часто используется вместе со статистическими функциями, например, сумма (`sum`), среднее (`avg`)
		`CREATE PROCEDURE Get_Values`
		`RETURNS (opSum FLOAT, opAvg FLOAT)`
		`AS`
		`BEGIN`
		`SELECT`
		`SUM(Salary),`
		`AVG(Salary)`
		`FROM Employee`
		`INTO :opSum,`
		`:opAvg`
		`END`
		В данном примере создается хранимая процедура `Get_Values`, в которой вычисляются общая сумма зарплаты и средняя зарплата для сотрудников некоторого предприятия (таблица `Employee`).

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример
		Полученные значения присваиваются выходным параметрам `opSum` и `opAvg`, которые возвращают данные значения в вызывающую программу. У рассматриваемой процедуры нет входных параметров

Оператор выбора нескольких записей
`FOR <Оператор выбора записи> DO <Оператор>;`
	Представляет собой оператор `SELECT`, который может возвращать несколько записей (либо ни одной).
	Оператор выбора нескольких записей может задавать не только отбор, но и обработку записей. После отбора записей согласно оператору `SELECT` для каждой из них выполняется оператор, указанный после слова `DO`
		`CREATE PROCEDURE Number`
		`RETURNS (opSum INTEGER)`
		`AS`
		`DECLARE VARIABLE`
		`n INTEGER;`
		`DECLARE VARIABLE`
		`x INTEGER;`
		`BEGIN`
		`x = 0;`
		`FOR SELECT`
		`_Value FROM`
		`_Table`

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример

		INTO n DO
		x = x + n;
		opSum = x;
		END
		
		Создается хранимая процедура Number, которая вычисляет сумму значений целочисленного столбца _Value таблицы _Table. Оператор выбора обеспечивает отбор всех записей таблицы, а оператор присваивания увеличивает значение суммы x на значение столбца _Value очередной записи. После перебора всех отобранных записей результат возвращается через выходной параметр opSum

Оператор возврата значений FOR <Оператор выбора записи> DO SUSPEND;	Оператор возврата значений отличается от оператора выбора нескольких записей тем, что после слова DO указывается оператор SUSPEND, который передает в вызывающее приложение или хранимую процедуру значения выходных параметров.	CREATE PROCEDURE Value_2 (ipValueMin FLOAT, ipValueMax FLOAT) RETURNS (opSurname VARCHAR(20), opValue FLOAT) AS

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример
	Обычно оператор SUSPEND используется в операторе SELECT выбора нескольких записей хранимой процедуры. Совместное использование операторов выбора нескольких записей и возврата значений обеспечивает фактически возврат в приложение совокупности записей, представляющих собой набор данных	BEGIN FOR SELECT Surname, _Value FROM Employee WHERE _Value >= :ipValueMin AND _Value <= :ipValueMax INTO :opSurname, :opValue DO SUSPEND END Создается хранимая процедура Value_2, которая получает набор данных. Совокупность записей этого набора данных образуют записи таблицы SLUGHASHIE сотрудников организации, у которых оклад находится в заданном диапазоне. Границы диапазона определяют входные параметры ipValueMin и ipValueMax. Набор данных включает столбцы Surname, _Value

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример

Оператор выхода из процедуры
`EXIT;`
	Прекращает выполнение хранимой процедуры и осуществляет передачу управления в вызывающую программу или процедуру. Если оператор выхода отсутствует, то завершение процедуры и передача управления в вызывающую программу или процедуру происходят сразу же после выполнения последнего оператора процедуры	`CREATE PROCEDURE Div (ipP FLOAT, ipV FLOAT)`
		`RETURNS (opResult FLOAT)`
		`AS`
		`BEGIN`
		`IF (pV = 0)`
		`THEN`
		`BEGIN pResult = 0;`
		`EXIT`
		`END`
		`opResult = :ipP / :ipV;`
		`END`
		
		В данном случае создается хранимая процедура `Div`, которая выполняет деление двух чисел. Если делитель равен нулю, то результат обнуляется и выполнение данной процедуры заканчивается

Таблица 5.1 *(продолжение)*

Оператор	Описание	Пример

Оператор вызова процедуры
`EXECUTE PROCEDURE Имя_процедуры [(<Список входных параметров>)]`
`RETURNING_VALUES (<Список выходных параметров>)`
	Предназначен для вызова другой хранимой процедуры. Оператор `EXECUTE` вызывает хранимую процедуру с указанным именем, входными и выходными параметрами. В частном случае у вызываемой процедуры параметры могут отсутствовать	`CREATE PROCEDURE Call`
		`RETURNS (opResult FLOAT)`
		`AS`
		`BEGIN`
		`EXECUTE PROCEDURE Div (30, 5) RETURNING_VALUES (opResult);`
		`END`
		
		Создается хранимая процедура `Call`, вызывающая ранее созданную процедуру деления двух чисел `Div`. В качестве входных параметров для деления передаются числа 30 и 5. Результат, возвращаемый процедурой `Div`, присваивается выходному параметру `opResult` процедуры `Call`

Таблица 5.1 *(окончание)*

Оператор	**Описание**	**Пример**
Оператор посылки сообщения `POST EVENT "имя_события";`	Предназначен для рассылки некоторого сообщения приложениям, связанным с сервером. Сообщение посылается всем приложениям при наступлении на сервере определенного события. Следует заметить, что для получения соответствующего сообщения приложение должно предварительно зарегистрироваться на сервере, для чего приложению необходимо использовать соответствующий компонент	`POST EVENT "STOP";` В данном случае приложениям, связанным с сервером, посылается сообщение `STOP`

В зависимости от того, сколько строк получается в качестве результата, можно выделить следующие виды хранимых процедур:

- хранимые процедуры, возвращающие одну строку;
- хранимые процедуры, возвращающие несколько строк.

Процедуры, которые возвращают одну строку, практически аналогичны процедурам языка Pascal и обеспечивают возврат значений выходных параметров. Такие хранимые процедуры также называют *процедурами действия*.

Процедуры, возвращающие несколько строк результатов, выполняют возврат набора данных, в котором каждая строка результатов является записью этого набора данных, и называются *процедурами выбора*.

Хранимая процедура выбора, как правило, вызывается с помощью оператора выбора `SELECT`, внутри которого размещается вызов процедуры.

Пример. Вызывать хранимую процедуру `Value_2` (см. табл. 5.1), выводящую фамилии и оклады сотрудников, значение оклада которых принадлежит заданному диапазону.

```
SELECT * FROM Value_2 (5000, 9000);
```

Функции

Как правило, в языке SQL имеется небольшое число встроенных функций. При необходимости пользователь может создать и использовать функции, которые нужно часто применять.

Функция, определяемая пользователем, представляет собой обычную функцию, написанную на алгоритмическом языке (например, Pascal), которая сохраняется в виде динамической библиотеки DLL и может быть вызвана обычным способом. В общем случае библиотека содержит несколько функций.

Для обеспечения вызова функции ОС Windows необходимо задать путь к соответствующей библиотеке.

Достоинствами использования функций, созданных пользователями, являются:

- расширение состава функций языка SQL;
- возможность использования функций другими приложениями.

Для употребления функции, определяемой пользователем, необходимо выполнить следующие действия:

1. Создать функцию и включить ее в соответствующую библиотеку (при необходимости создать также библиотеку — модуль DLL);
2. Объявить функцию на сервере;
3. Вызвать функцию в операторе SQL.

Пример. Использование функции, выполняющей следующие действия: извлечение квадратного корня из числа и прибавление квадрата числа (для чисел больших либо равных нулю). Имя функции — `Action`, имя библиотеки — `MyLibrary`.

Решение.

В среде Delphi код библиотеки выглядит следующим образом:

```
library MyLibrary;
uses
SysUtils;
Classes;
   function Action (x: real): real; cdecl; export;
    begin
      if x<0 then Action:=0
       else Action:=sqrt(x) + x*x
    end;
exports Action;
begin end.
```

В заголовке данной функции указаны слова `cdecl` и `export`, определяющие соглашения для параметров и назначение функции. Имя экспортируемой функции указывается в разделе `exports`.

Модуль библиотеки `MyLibrary`, который получен в результате трансляции, необходимо скопировать в каталог, откуда он будет доступен ОС Windows (например, C:\WINDOWS\SYSTEM).

Объявление функции пользователя на сервере выполняется следующим образом:

```
DECLARE EXTERNAL FUNCTION Имя_функции [<Тип данных>, ... <Тип данных>]

RETURNS (<Тип данных> [BY VALUE] | CSTRING (<Целое число)} ENTRY_POINT "Зарегистрированное_имя_функции" MODULE_NAME "Имя_библиотеки";
```

Таким образом, имеем:

```
DECLARE EXTERNAL FUNCTION Action FLOAT RETURNS FLOAT BY VALUE ENTRY_POINT "Action" MODULE_NAME "MyLibrary";
```

Для отмены объявления функции на сервере (удаления) необходимо воспользоваться оператором `DROP`:

```
DROP EXTERNAL FUNCTION <Имя функции>;
```

После того как пользовательская функция объявлена, ее можно использовать в операторах языка SQL наряду со встроенными функциями. Так, вызов функции `Action` в операторе отбора записей будет иметь следующий вид:

```
SELECT *

FROM Graphs

WHERE CoordY <= Action (CoordX);
```

Данный оператор `SELECT` выбирает из таблицы `Graphs` все записи, для которых значение столбца `CoordY` не превосходит суммы квадратного корня и квадрата числа из значения столбца `CoordX`.

Замечание

Локальная версия сервера InterBase не поддерживает работу с пользовательскими функциями.

Восстановление базы данных

Восстановление системы базы данных — это возможность возвращения базы данных в правильное состояние при любом сбое в работе системы, который сделал неправильным или подозрительным текущее состояние базы данных. Основной принцип, на котором строится восстановление, — *принцип избыточности*. Таким образом, если любая часть информации, которая содержится в базе данных, может быть восстановлена из другой хранимой в системе избыточной информации, значит, база данных восстанавливаема.

Транзакции

Для процесса восстановления существенное значение имеют транзакции. *Транзакция* — действие или серия действий, выполняемых пользователем или прикладной программой как единое целое, которые осуществляют доступ или изменение содержимого базы данных.

В реляционных базах данных транзакции осуществляются с помощью команд DML (`INSERT`, `UPDATE` и `DELETE`). Транзакция представляет собой внесение некоторых изменений в базу данных.

Пример. Для приема заказа от клиента программа ввода заказов должна (см. рис. 4.5) осуществить следующие действия.

1. Выполнить запрос к таблице `Products` и проверить наличие товара на складе.
2. Добавить заказ в таблицу `Orders`.

3. Обновить таблицу `Products`: вычесть заказанное количество товара из количества товара, которое имеется в наличии.
4. Обновить таблицу `Employee`, добавив стоимость заказа к объему продаж служащего, принявшего заказ.
5. Обновить таблицу `Offices`, добавив стоимость заказа к объему продаж офиса, в котором работает данный служащий.

В стандарте ANSI/ISO определена модель транзакций. Транзакция автоматически начинается с выполнения пользователем или программой первой инструкции SQL. Далее происходит последовательное выполнение остальных инструкций SQL одним из двух способов: `COMMIT` или `ROLLBACK`.

Существуют две модели транзакций:

- автоматическое выполнение транзакций (т. е. инструкция `COMMIT` выполняется автоматически после каждой инструкции SQL);
- транзакции, использующие точки сохранения, не выполняются автоматически; их выполнение начинается с инструкции `BEGIN TRANSACTION`.

Транзакции в реляционной СУБД подчиняются следующему правилу:

"Инструкции, входящие в транзакцию, рассматриваются как неделимое целое: либо все инструкции выполняются успешно, либо ни одна из них не должна быть выполнена".

Транзакции обладают ACID-свойствами *(см. главу 1)*.

- *Атомарность*. Транзакции атомарны (выполняется все или ничего). Каждая транзакция имеет начало и конец.
- *Согласованность*. Транзакции согласованно защищают базу данных, т. е. транзакции переводят одно согласованное состояние базы данных в другое без обязательной поддержки согласованности во всех промежуточных точках.

- *Изоляция*. Транзакции отделены друг от друга: если запускается много конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется. Для двух любых отдаленных транзакций T_1 и T_2 справедливо утверждение: *T_1 сможет увидеть обновление T_2 только после выполнения T_2, а T_2 сможет увидеть обновление T_1 только после выполнения T_1.*
- *Долговечность*. Любую транзакцию можно либо сохранить, либо отметить, т. е. когда транзакция выполнена, ее обновления сохраняются, даже если в следующий момент произойдет сбой системы.

Управление транзакциями

Управление транзакциями обеспечивает возможность контроля транзакций, выполняемых в рамках общей системы управления базой данных.

Чтобы передать базе данных изменения, проведенные транзакцией, используется оператор `COMMIT`. При внесении в базу данных большого количества изменений рекомендуется использовать `COMMIT` как можно чаще.

Инструкция `ROLLBACK` отменяет транзакции, которые еще не были сохранены, и база данных возвращается в состояние, в котором она была до начала транзакции.

Инструкции `COMMIT` и `ROLLBACK` можно использовать в интерактивном режиме, хотя на практике это редко применяется. Во многих интерактивных СУБД по умолчанию установлен режим "автоматического завершения", при котором инструкция `COMMIT` выполняется после каждой инструкции SQL. Таким образом, каждая интерактивная инструкция SQL становится отдельной транзакцией.

В некоторых СУБД имеются дополнительные возможности при выполнении транзакций.

- Создание *точки отката*, т. е. создание такого места между операторами транзакции, к которому можно обратиться, не отменяя всю транзакцию:

```
SAVEPOINT имя_точки_отката;
```

  либо

```
SAVE TRANSACTION имя_точки_отката;
```

  Имя точки отката должно быть уникальным внутри соответствующей группы транзакций, но может совпадать с именем таблицы или другого объекта.

- Команда возврата к точке отката (сохранения):

```
ROLLBACK TO имя_точки_отката;
```

- Удаление точки отката (сохранения):

```
RELEASE SAVEPOINT имя_точки_отката;
```

Транзакция начинается с выполнения оператора `BEGIN TRANSACTION` и заканчивается успешным выполнением либо оператора `COMMIT`, либо `ROLLBACK`. Оператор `COMMIT` устанавливает точку фиксации (точку синхронизации — syncpoint), которая соответствует концу логической единицы работы, т. е. точке, в которой база данных находится (или будет находиться) в состоянии согласованности. Оператор `ROLLBACK` возвращает базу данных в предыдущую фиксацию.

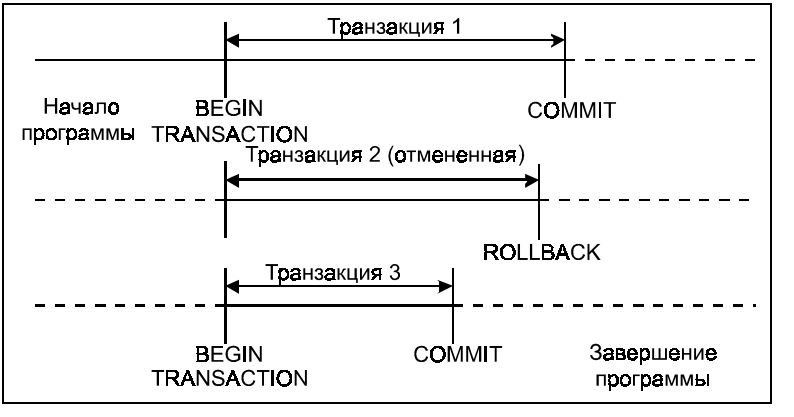


Рис. 5.1. Программа как согласованность транзакций

Таким образом, COMMIT и ROLLBACK завершают транзакцию. Следует учесть, что выполнение программы состоит из согласования нескольких транзакций, которые выполняются либо отменяются (рис. 5.1).

Журнал транзакций

В каждой СУБД существует журнал транзакций для фиксации всех инструкций, обращенных к базе данных. Как правило, журнал содержит две записи по строкам: до и после изменения строки. Только после записи в журнале СУБД изменяет физическую строку на диске. В случае системного сбоя администратор базы данных восстанавливает ее с помощью специальной утилиты восстановления, которая входит в комплект СУБД. Данная утилита просматривает журнал транзакций и ищет все незавершенные транзакции к моменту сбоя, после чего в базе данных происходят изменения согласно завершенным транзакциям и отмена незавершенных.

В результате использования журнала транзакций увеличиваются затраты времени, необходимые для внесения изменений в базу данных. Обычно в СУБД применяются сложные схемы ведения журнала транзакций, который, как правило, хранится на высокоскоростном жестком диске отдельно от базы данных.

Восстановление системы

Существуют два вида глобальных нарушений системы:

- *отказы системы* (аварийный отказ программного обеспечения) — затрагивают все выполняющиеся в данный момент транзакции, но физически не нарушают базу данных в целом;
- *отказы носителей* (аварийный отказ аппаратуры) — представляют угрозу для базы данных или какой-либо ее части и затрагивают те транзакции, которые используют данную часть базы данных.

Отказы системы

Критической точкой в отказе системы является потеря содержимого основной (оперативной) памяти, т. е. рабочих буферов базы данных. Так как точное состояние какой-либо выполняющейся в момент нарушения транзакции не известно, транзакция может не завершиться успешно и будет *отменена* при перезагрузке системы.

Выполнение и невыполнение транзакций отождествляется в СУБД через *запись контрольной точки*, которая включает в себя физическую запись содержимого рабочих буферов базы данных непосредственно в базе данных. При перезагрузке система сначала идентифицирует все транзакции, которые осуществлялись в промежуток времени между записью контрольной точки и отказом системы. При этом выполняются следующие действия:

1. Создание двух списков транзакций: `UNDO` (отменить) и `REDO` (выполнить повторно). В `UNDO` заносятся все транзакции, зафиксированные контрольной точкой. Список `REDO` некоторое время остается пустым.
2. Поиск в файле регистрации (журнале), начиная с записи контрольной точки.
3. Если в файле регистрации обнаружена запись `BEGIN TRANSACTION` (начало транзакции), то эта транзакция добавляется в список `UNDO`.
4. Если в файле регистрации обнаружена запись `COMMIT` (конец транзакции), то эта транзакция добавляется в список `REDO`.
5. Когда достигается конец файла регистрации, списки `UNDO` и `REDO` анализируются для идентификации транзакций, появившихся в списке `REDO`, и транзакций, оставшихся в списке `UNDO`.
6. Далее система "просматривает назад" файл регистрации, отменяя транзакции из списка `UNDO`, а затем, "просматривая снова вперед", повторно выполняет транзакции из списка `REDO`.

Восстановление базы данных в правильное состояние путем отмены выполненных операций иногда называется *обратным восстановлением*. Восстановление базы данных в правильное состояние повторным выполнением называется *прямым восстановлением*.

Пример. Пусть в некоторое время $T_к$ произведена запись контрольной точки, которая зафиксировала выполнение транзакций T_2, T_3 и T_7. (рис. 5.2). Спустя некоторое время в системе стали выполняться транзакции T_4, T_5 и T_6. В момент $T_о$ произошел аварийный отказ системы. После перезагрузки системы и идентификации всех транзакций, произошедших в период между фиксацией контрольной точки и отказом системы, транзакции T_2, T_4, T_6 и T_7 выполнены повторно, а транзакции T_3 и T_5 отменены.

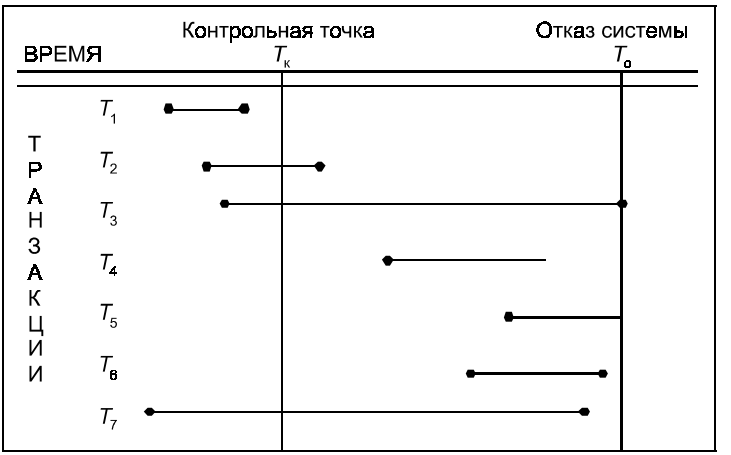


Рис. 5.2. Транзакции и отказ системы

Отказы носителей

Отказы носителей представляют собой нарушения типа поломки головок дискового накопителя или дискового кон-

троллера, когда некоторая часть базы данных разрушается физически. Восстановление после такого нарушения включает перезапись базы данных с резервной копии (*дампа*) и последующее использование файла регистрации — как его активной, так и его архивной части, т. е. повторное выполнение всех транзакций, вызванное применением резервной копии.

Параллелизм в базах данных

Процесс организации одновременного выполнения в базе данных различных транзакций с доступом к одним и тем же данным, который гарантирует исключение их взаимного влияния друг на друга, называется *управлением параллельностью*.

Для корректной обработки параллельных транзакций без возникновения конфликтных ситуаций необходимо использовать некоторый *метод управления параллелизмом*.

Проблемы параллелизма. Транзакции в многопользовательском режиме

В идеальном случае каждый пользователь должен работать с базой данных так, как если бы он имел к ней монопольный доступ.

Когда две транзакции, T_1 и T_2, выполняются параллельно, СУБД гарантирует, что результаты их выполнения будут точно такими же, как и в случае, если:

- вначале выполняется транзакция T_1, затем транзакция T_2;
- вначале выполняется транзакция T_2, а затем транзакция T_1.

Данная концепция называется *сериализацией* транзакций.

Перечислим основные проблемы, возникающие при параллельной обработке транзакций.

- *Проблема пропавшего обновления* — двое пользователей обращаются к базе данных, каждый из них обновляет одну и ту же таблицу одновременно. Таким образом, пользователи извлекают из базы данных одни и те же данные, используют их для каких-либо расчетов, а затем пытаются обновить эти данные.
- *Проблема промежуточных данных* — некоторая транзакция имеет доступ к промежуточным данным, которые в текущий момент также используются другой транзакцией. Реально первая транзакция обращается к данным, не предполагая, что они промежуточные при работе второй транзакции. Таким образом, есть вероятность того, что обновление значений базы данных не будет завершено никогда.
- *Проблема несовместимого анализа (несогласованных данных)* — использование данных, хранящихся в базе данных, одновременно различными транзакциями для анализа данных может привести к неверным результирующим значениям. Часто несогласованность данных приводит к появлению так называемых строк-призраков.

Для обработок параллельных транзакций практически во всех СУБД применяется довольно сложный механизм блокировки, который включается/отключается в той либо иной транзакции.

Блокировка

Блокировка сводится к следующим действиям. Когда транзакция T_1 обращается к базе данных, СУБД автоматически блокирует все части базы данных, в которых транзакция осуществляет выборку или изменение. Транзакция T_2 выполняется параллельно, и СУБД также блокирует те части базы данных, к которым она обращается. Если транзакция T_2 обращается к той части базы данных, которая заблокирована транзакцией

T_1, то СУБД приостанавливает выполнение транзакции T_2, заставляя ее ждать до тех пор, пока данные не будут разблокированы. СУБД снимает блокировку, вызванную транзакцией T_1, только после того, как в той транзакции встретится инструкция `COMMIT` либо `ROLLBACK`. Затем СУБД позволяет продолжить выполнение транзакции T_2. Теперь транзакция T_2 блокирует эту же часть базы данных, защищая ее от других транзакций (рис. 5.3).

Рис. 5.3. Блокировка при одновременном выполнении двух транзакций

Уровни блокировки

В базе данных блокировка может быть реализована на различных уровнях.

- *Блокировка всей базы данных* — грубая форма блокировки: легко реализуется; в каждый момент выполняется только одна транзакция.

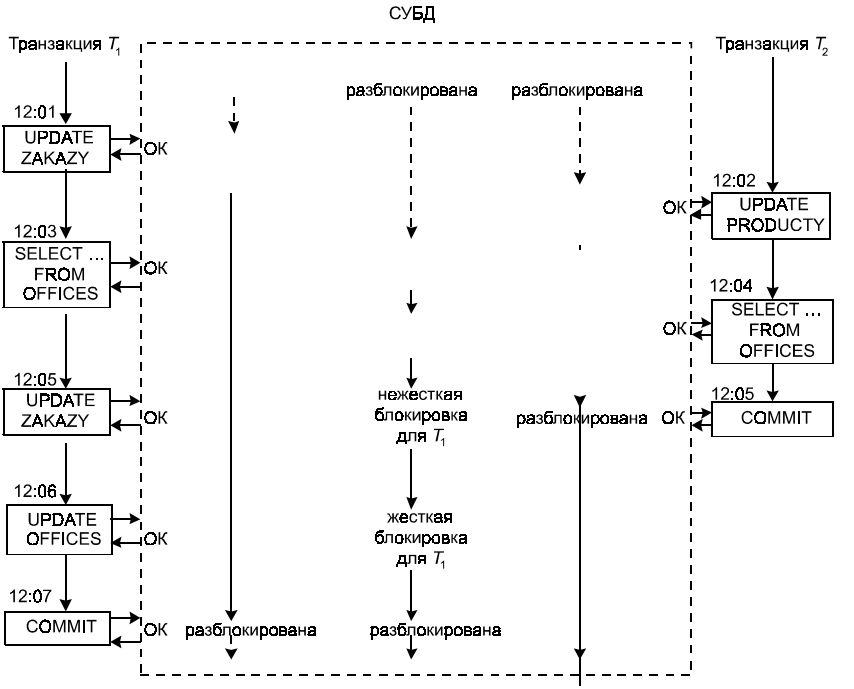


Рис. 5.4. Жесткая и нежесткая блокировки

- *Блокировка на уровне таблиц* — блокировка только тех таблиц, к которым обращается транзакция. Остальные транзакции могут обращаться к другим таблицам. Если в программах несколько пользователей одновременно об-

ращаются к одним и тем же таблицам (ввод заказов), то данный вид блокировки приводит к низкой производительности.

- *Блокировка на уровне страниц* — блокировка СУБД отдельных блоков данных на диске в тот момент, когда транзакция обращается к ним.
- *Блокировка на уровне строк* — допускает выполнение параллельных транзакций, которые обращаются к двум различным строкам таблицы, даже если эти строки содержатся на одной странице.

Типы блокировки (рис. 5.4).

- *Нежесткая* блокировка используется в транзакциях на извлечение информации из базы данных. Параллельно могут работать другие транзакции.
- *Жесткая* блокировка применяется в транзакциях на обновление информации в базе данных. Если транзакция жестко заблокировала какие-нибудь данные, другие транзакции не могут обращаться к ним ни для выборки, ни для записи.

Допустимые комбинации блокировок для двух параллельно выполняемых транзакций

Следует отметить, что транзакция может применять для данных жесткую блокировку только тогда, когда ни одна другая транзакция не блокирует эти данные. Если транзакция пытается осуществить блокировку, не разрешенную правилами (табл. 5.2), ее выполнение приостанавливается до тех пор, пока другие транзакции не разблокируют необходимые ей данные.

Таблица 5.2. *Правила применения жесткой и нежесткой блокировок*

		Транзакция T_2		
		Разблокировка	**Нежесткая блокировка**	**Жесткая блокировка**
Транзакция T_1	**Разблокировка**	Да	Да	Да
	Нежесткая блокировка	Да	Да	Нет
	Жесткая блокировка	Да	Нет	Нет

Тупиковые ситуации

Использование любого механизма блокировок для параллельного выполнения транзакций приводит к так называемой тупиковой ситуации (рис. 5.5).

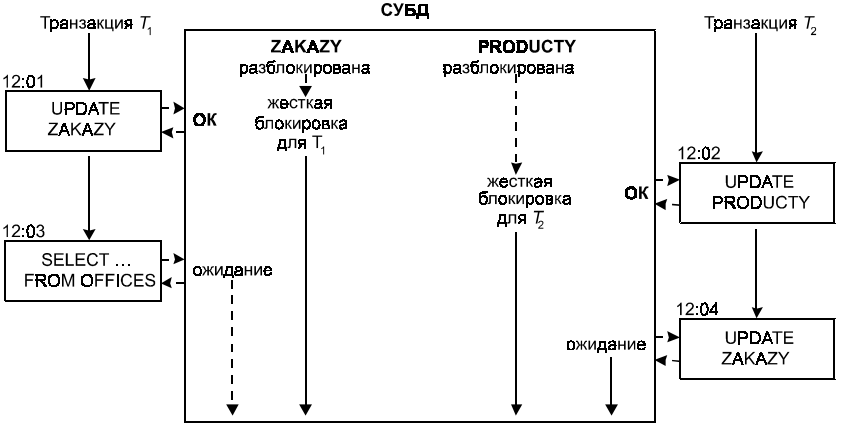


Рис. 5.5. Тупиковая ситуация двух транзакций

Если СУБД обнаруживает "тупик", то произвольно выбирает одну транзакцию в качестве отбрасываемой и отменяет ее. Это означает, что любая инструкция SQL может вернуть код ошибки. Однако такая схема на практике — лучший выход из тупиковых ситуаций.

Усовершенствованные методы блокировки

Явная блокировка

Блокирует целую таблицу или другую часть базы данных, если необходимо к ней обращаться многократно.

В некоторых СУБД для явной блокировки целой таблицы применяется следующая инструкция:

```
LOCK TABLE имя_таблицы IN (SHARE || EXCLUSIVE) MODE
```

Режим `EXCLUSIVE` осуществляет жесткую блокировку всей таблицы. Данный режим следует использовать в том случае, если транзакция выполняет пакетное обновление всех строк таблицы.

Режим `SHARE` осуществляет нежесткую блокировку таблицы. Режим следует использовать тогда, когда необходимо получить "мгновенный список" таблицы в какой-то определенный момент времени.

Уровни изоляции

Если в течение транзакции программа выполняет запрос на выборку информации из базы данных, потом выполняется другое действие, а затем выполняется опять тот же запрос на выборку, то СУБД гарантирует, что данные, возвращенные запросами, будут идентичными (исключая случай изменения данных самой транзакцией). Возможность повторной выборки во время транзакции означает наиболее высокую степень изоляции программы от других программ и пользователей.

Степень изоляции одной транзакции от других транзакций называется *уровнем изоляции*.

Инструкция `SET TRANSACTION` используется для установки уровня изоляции текущей транзакции (рис. 5.6).

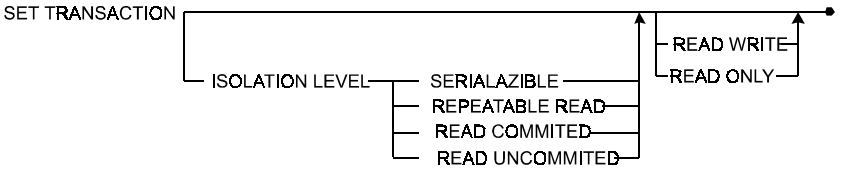


Рис. 5.6. Диаграмма инструкции `SET TRANSACTION`

В данной инструкции применяется несколько уровней изоляции (`ISOLATION LEVEL`).

- `SERIALIZABLE` — самый высокий уровень изоляции. Гарантирует, что результаты параллельного выполнения транзакций будут точно такими же, как если бы эти транзакции выполнялись последовательно. Данный уровень изоляции устанавливается по умолчанию.
- `REPEATABLE READ` — второй по степени изоляции уровень. На этом уровне транзакция не имеет доступа к промежуточным или окончательным результатам других транзакций, выполняющих обновление данных, поэтому такие проблемы, как пропавшее обновление, промежуточные или несогласованные данные, возникнуть не могут. Однако во время своей транзакции можно увидеть строку, добавленную в базу другой транзакцией. Поэтому один и тот же запрос к нескольким строкам, выполненный дважды в течение одной транзакции, может возвратить различные таблицы результатов (проблема строк-призраков).
- `READ COMMITTED` — третий уровень по степени изоляции. Транзакция не имеет доступа к промежуточным результатам других транзакций, поэтому не возникают проблемы

пропавшего обновления и промежуточных данных. Однако окончательные результаты других параллельно выполняемых транзакций могут быть доступны этой транзакции.

- `READ UNCOMMITTED` — самый низкий уровень изоляции. В этом режиме на выполнение транзакции могут повлиять как окончательные, так и промежуточные результаты других транзакций, поэтому могут возникнуть проблемы строк-призраков, промежуточных и несогласованных данных.

В инструкции `SET TRANSACTION` можно также указать, какого типа операции осуществляет транзакция:

- `READ ONLY` — осуществление только инструкций на выборку информации из базы данных;
- `READ WRITE` — выполнение инструкций как на выборку, так и на изменение данных.

По умолчанию устанавливается уровень изоляции `SERIALIZABLE`. Если задан уровень `READ UNCOMMITTED`, то предполагается, что транзакция имеет атрибут `READ ONLY`, поэтому `READ WRITE` задавать нельзя. В остальных случаях по умолчанию считается, что транзакция имеет атрибут `READ WRITE`. Установки по умолчанию обеспечивают максимальную безопасность транзакций, хотя из-за этого возможны потери производительности.

Параметры блокировки

Администратор базы данных может вручную установить размер блокируемого участка базы данных и другие параметры блокировки.

- *Размер заблокированного участка.* В каждой СУБД можно определить участок блокировки — таблица, страница, строка или другие участки данных. Для различных прикладных программ подходящими будут различные размеры заблокированных участков.

- *Число блокировок.* Обычно СУБД позволяет каждой транзакции иметь ограниченное число блокировок. Предел устанавливает администратор базы данных, увеличивая его для сложных транзакций и уменьшая для простых.
- *Наращивание блокировок.* Часто СУБД автоматически наращивает блокировки, заменяя множество маленьких блокировок одной большой (например, несколько страничных блокировок заменяется блокировкой таблицы). Администратор базы данных также может управлять процессом наращивания.

Интервал блокировки

Если транзакция заблокирована другой транзакцией, она может долго ожидать снятие блокировки второй транзакцией. Потому в некоторых СУБД реализованы интервалы блокировки. При использовании таких интервалов инструкция SQL завершается неуспешно и возвращает код ошибки SQL, если не может установить требуемые блокировки в течение определенного промежутка времени. Этот промежуток обычно задается администратором базы данных.

Упорядоченность транзакций

Для заданного набора транзакций любой порядок их выполнения называется *графиком запуска*. Выполнение транзакций по одной без их чередования называется последовательным графиком запуска, а непоследовательное — чередующимся графиком. Два графика будут *эквивалентны*, если при их выполнении будет получен одинаковый результат независимо от исходного состояния базы данных.

При выполнении двух различных последовательных графиков запуска, содержащих одинаковый набор транзакций, можно получить совершенно различные результаты, т. е. выполнение двух различных чередующихся графиков запуска с одинаковыми транзакциями может также привести к различным результатам, которые могут быть восприняты как верные.

Концепция способности к упорядоченности была впервые предложена Есвараном (1976), который сформулировал теоретические основы параллелизма и доказал теорему двухфазной блокировки:

Если все транзакции подчиняются "протоколу двухфазной блокировки", то для всех возможных чередующихся графиков запуска существует возможность упорядочения.

В свою очередь, *протокол двухфазной блокировки* формулируется следующим образом:

- перед выполнением каких-либо операций с некоторым объектом (например, с кортежем базы данных) транзакция должна заблокировать данный кортеж;
- после снятия блокировки транзакция не должна накладывать никаких других блокировок.

Таким образом, транзакция, которая подчиняется этому протоколу, характеризуется *фазой наложения блокировки* и *фазой снятия блокировки*. На практике вторая фаза часто сводится к единственной операции окончания выполнения в конце транзакции.

Если T_1 и T_2 являются любыми двумя транзакциями некоторого графика запуска, допускающего возможность упорядочения, то либо T_1 логически предшествует T_2, либо T_2 логически предшествует T_1, т. е. *либо T_2 использует результаты выполнения транзакции T_1, либо T_1 использует результаты выполнения транзакции T_2* (если транзакция T_1 приводит к обновлению кортежей (r_1, r_2, …, r_n) и транзакция T_2 применяет эти кортежи в качестве входных данных, то используются либо все обновленные с помощью T_1 кортежи, либо полностью не обновленные кортежи до выполнения транзакции T_1).

Если некоторая транзакция T_1 не является двухфазной (т. е. не удовлетворяет протоколу двухфазной блокировки), то всегда можно построить некоторую другую транзакцию T_2, которая при чередующемся выполнении вместе с транзакцией T_1 может привести к неправильному графику запуска, не подлежа-

щему упорядочению. В настоящее время с целью понижения требований к ресурсам и, следовательно, повышения производительности и пропускной способности в реальных системах обычно предусмотрено использование не двухфазных транзакций, а транзакций с "ранним снятием блокировки" (еще до выполнения операции прекращения транзакций) и наложением нескольких блокировок. Однако применение таких транзакций связано с большим риском, т. к. при использовании недвухфазной транзакции T_1 предполагается, что в данной системе не существует никакой другой чередующейся с ней транзакции T_2 (в противном случае в системе возможно получение ошибочных результатов).

Администрирование баз данных

Защита базы данных

Данные представляют собой ценный ресурс, доступ к которому необходимо строго контролировать и регламентировать. Под *защитой* данных понимается обеспечение защищенности базы данных от несанкционированного доступа, произошедшего с помощью различных компьютерных и некомпьютерных средств. Защита базы данных должна охватывать используемое оборудование, программное обеспечение, персонал и собственно данные. Защита базы данных предполагает предотвращение таких нарушений, как:

- похищение и фальсификация данных;
- утрата конфиденциальности (нарушение тайны) и нарушение неприкосновенности личных данных;
- утрата целостности и потеря доступности данных;
- физическое повреждение оборудования;
- ситуации, связанные со стихийными бедствиями;
- внедрение компьютерных вирусов;

- электронные наводки и радиация;
- проблемы, возникающие с персоналом (нехватка персонала, недостаточная обученность, забастовки и т. п.).

Под *опасностью* понимается любая ситуация или событие, намеренное или непреднамеренное, которые способны неблагоприятно повлиять на систему, а следовательно, и на всю организацию.

Уровень потерь, понесенных организацией в результате реализации некоторой угрозы, зависит от многих причин, включая наличие заранее продуманных контрмер и планов для преодоления непредвиденных ситуаций. Продолжительность периода бездействия и быстрота восстановления базы данных зависят от следующих факторов:

- наличия в системе резервного оборудования с развернутым программным обеспечением (ПО);
- времени последнего резервного копирования системы;
- времени, необходимого на восстановление системы;
- возможности восстановления и ввода в эксплуатацию разрушенных данных.

В любой организации должны быть установлены типы возможных опасностей, которым может подвергнуться ее компьютерная система, и разработаны соответствующие планы, а также требуемые контрмеры с оценкой уровня затрат, необходимых для их реализации.

Для предотвращения опасностей, угрожающих компьютерным системам, существует ряд контрмер различных типов, включающих как физические наблюдения за системой, так и разнообразные административно-организационные процедуры. Общий уровень защищенности СУБД определяется возможностями используемой операционной системы. Среди *компьютерных средств контроля* можно выделить следующие:

- авторизация пользователей;
- представления;

- резервное копирование и восстановление;
- поддержка целостности;
- шифрование;
- вспомогательные процедуры.

Обычно для платформы IBM PC применяются не все перечисленные типы средств контроля.

Авторизация пользователей — это предоставление прав (привилегий), позволяющих их владельцу иметь законный доступ к системе или к ее объектам. Средства авторизации пользователей могут быть встроены непосредственно в ПО и управлять не только предоставленными пользователям правами доступа к системе или объектам, но и наборами операций, которые пользователи могут выполнять с каждым доступным объектом. Процесс авторизации включает аутентификацию субъектов, требующих получения доступа к объектам.

Аутентификация — механизм определения того, является ли пользователь тем, за кого себя выдает. Использование паролей является наиболее распространенным методом аутентификации пользователей. Однако этот подход не дает полной гарантии того, что пользователь является именно тем, за кого себя выдает. Как только пользователь получает право доступа к СУБД, ему могут предоставляться различные привилегии. Некоторые СУБД функционируют как *закрытые системы*, в которых пользователям кроме разрешения на доступ к самой СУБД требуется иметь отдельные разрешения на доступ к конкретным ее объектам (выдаются администратором базы данных). *Открытые системы* по умолчанию предоставляют авторизированным пользователям полный доступ ко всем объектам БД.

Представления — это виртуальные (реально не существующие в базе данных, *см. главу 4*) таблицы, которые создаются по требованию отдельного пользователя в необходимый момент времени.

Резервное копирование — периодически выполняемая процедура получения копии базы данных и ее файла журнала (а также, возможно, программ) на носителе, сохраняемом отдельно от системы. Процедура создания и обслуживания файла журнала, содержащего сведения обо всех изменениях, внесенных в базу данных с момента создания последней резервной копии, и предназначенного для обеспечения эффективного восстановления системы в случае ее отказа, называется *ведением журнала*.

Поддержка целостности предполагает предотвращение перехода данных в несогласованное состояние, т. е. угрозы получения ошибочных или некорректных результатов расчета.

Шифрование — кодирование данных с использованием специального алгоритма, в результате чего данные становятся недоступными для чтения любой программой, не имеющей ключа дешифрования. Для организации защищенной передачи данных по незащищенным сетям должны использоваться системы *шифрования*, которые включают:

- ключ шифрования, предназначенный для шифрования исходных данных (обычного текста);
- алгоритм шифрования, который описывает, как с помощью ключа шифрования преобразовать обычный текст в шифротекст;
- ключ дешифрования, предназначенный для дешифрования шифротекста;
- алгоритм дешифрования, который описывает, как с помощью ключа дешифрования преобразовать шифротекст в исходный обычный текст.

Вспомогательные процедуры нацелены, прежде всего, на разработку новых алгоритмов для проведения описанных ранее возможностей компьютерных средств контроля. Сюда можно отнести процедуры авторизации и аутентификации; процедуры, контролирующие процессы создания резервных копий; процедуры восстановления; установка нового прикладного ПО, установка или модернизация системного ПО; *аудит* —

проверка того, все ли предусмотренные средства управления задействованы и соответствует ли уровень защищенности установленным требованиям.

Аудиторская проверка предусматривает контроль следующих используемых процедур:

- поддержание точности вводимых данных;
- поддержание точности процедур обработки данных;
- предотвращение появления и своевременное обнаружение ошибок в процессе выполнения программ;
- корректное тестирование, документирование и сопровождение разработанных программных средств;
- предупреждение несанкционированного изменения программ;
- предоставление прав доступа и контроль за их использованием;
- поддержание документации в актуальном состоянии.

Некомпьютерные средства контроля

Некомпьютерные средства контроля включают такие методы, как выработку ограничений, соглашений и других административных мер, не связанных с компьютерной поддержкой.

- Меры обеспечения безопасности и планирование защиты от непредвиденных обстоятельств (область деловых процессов организации, для которых они устанавливаются; ответственность и обязанности отдельных работников; дисциплинарные меры, принимаемые в случае обнаружения нарушения установленных ограничений; процедуры, которые должны обязательно выполняться).
- Контроль за персоналом.
- Защита помещений и хранилищ.
- Гарантийные соглашения.

- Договоры о сопровождении.
- Контроль за физическим доступом.

Обычно основными аспектами защиты БД, на которые необходимо обращать внимание администратору БД, являются безопасность и целостность баз данных.

Итак, как упоминалось ранее, *безопасность* (защита данных от несанкционированного доступа и изменения/разрушения данных) — предоставляет определенным пользователям выполнять некоторые действия над базой данных. *Целостность* — подразумевает корректное выполнение разрешенных пользователям действий.

Безопасность

В современных СУБД поддерживается один из двух подходов к вопросу обеспечения безопасности данных: избирательный либо обязательный.

Избирательный подход обеспечивает пользователю различные права (привилегия или полномочия) при работе с разными объектами базы данных. Более того, разные пользователи обычно обладают разными правами доступа к одному и тому же объекту. Поэтому избирательные схемы характеризуются значительной гибкостью.

Обязательный подход каждому объекту базы данных присваивает некоторый *классификационный* уровень, а каждому пользователю обеспечивается некоторый уровень *допуска*. При таком подходе доступом к определенному объекту обладают только пользователи с соответствующим уровнем допуска. Обязательные схемы достаточно узки и статичны.

Все решения насчет выбора того или иного подхода к обеспечению прав доступа принимаются на стратегическом, а не на техническом уровне.

- Результаты стратегических решений должны быть известны системе (выполнены на основе утверждений, заданных с помощью необходимого языка) и сохраняться в ней (сохра-

нение их в каталоге в виде правил безопасности — полномочий).

- Наличие средств регулирования запросов доступа по отношению к соответствующим правилам безопасности (запрос доступа — комбинация запрашиваемой операции, запрашиваемого объекта и запрашивающего пользователя). Такая проверка выполняется подсистемой безопасности СУБД, которая также называется подсистемой полномочий.
- В системе должны быть предусмотрены способы опознания источника запроса, т. е. опознания запрашивающего пользователя. Как правило, это идентификатор пользователя и пароль. В системе могут поддерживаться группы пользователей и обеспечиваться одинаковые права доступа для пользователей одной группы. Кроме того, операции добавления отдельных пользователей в группу или их удаления из нее могут выполняться независимо от операции задания привилегий для этой группы (местом хранения информации о принадлежности к группе также является системный каталог или, возможно, БД).

Избирательное управление доступом

В общем случае правила безопасности обладают пятью компонентами и задаются следующими директивами:

```
CREATE SECURITY RULE имя_правила
GRANT список_привилегий_через_запятую
ON выражение
TO список_пользователей_через_запятую
[ON ATTEMPTED VIOLATION действие];
```

`Имя_правила` — имя нового правила безопасности. Правило будет зарегистрировано в системном каталоге под этим именем. Это имя также, вероятно, появится в любом системном сообщении или сообщении о состоянии системы в ответ на нарушение этого правила.

В качестве привилегий можно использовать одну из перечисленных далее:

```
RETRIEVE [(список_атрибутов_через_запятую)]
INSERT
UPDATE [(список_атрибутов_через_запятую)]
DELETE
ALL
```

Здесь RETRIEVE — привилегия извлечения, INSERT — вставки, UPDATE — обновления, DELETE — удаления. Привилегия RETRIEVE необходима для *упоминания* соответствующего объекта, например, в определении представления или ограничении целостности так же, как и для *самого* извлечения. Если вместе с привилегией RETRIEVE задан список атрибутов, то она применяется только к заданным атрибутам. Привилегия UPDATE со списком атрибутов определяется аналогично. Спецификация ALL является сокращенной записью использования всех привилегий — RETRIEVE, INSERT, UPDATE, DELETE.

Обозначение выражение включает выражение реляционного исчисления (или реляционной алгебры), в котором задан диапазон данного правила. Один из объектов списка в этом выражении должен точно идентифицировать одну переменную диапазона, соответствующую точно одному отношению.

Пользователь служит идентификатором некоторого пользователя, известного системе. Для него также может быть задана директива ALL (для всех известных пользователей).

Действие включает процедуру произвольной сложности. По умолчанию принимается отказ в выполнении запрашиваемого действия. Однако в сложных ситуациях необходимо применить другое действие (например, потребовать завершить выполнение программы или заблокировать клавиатуру пользователя, либо записать все произошедшие нарушения правил безопасности в специальном файле регистрации, так называемое "отслеживание угроз"). В системе должен быть предусмотрен способ устранения существующих правил безопасности:

```
DESTROY SECURITY RULE правило ;
```

Пример. Создать правило безопасности, позволяющее извлечение и удаление данных (S# (номер), SURNAME (фамилия), CITY (город)), касающихся поставщиков, находящихся вне Москвы, отношения _TABLE, для пользователей ANDREW, IRINA, SERGE. При нарушении данного правила не выполнять никаких действий.

Решение.

```
CREATE SECURITY RULE RULE_1
/* Создать правило безопасности RULE_1 */
      CRANT RETRIEVE (S#, SURNAME, CITY), DELETE
      /* Позволить извлечение (S#, SURNAME, CITY)
       и удаление */
      ON _TABLE WHERE _TABLE.CITY <> 'Moscow'
      /* Для поставщиков не из Москвы отношения _TABLE */
      TO ANDREW, IRINA, SERGE
      /* Для пользователей: Андрей, Ирина, Сергей */
      ON ATTEMPTED VIOLATION REJECT;
      /* Не выполнять при нарушении этого правила */
```

Пример. Создать правило безопасности, позволяющее извлечение столбцов S# (номер), SURNAME (фамилия), CITY (город) отношения _TABLE для пользователей ANDREW, IRINA, SERGE.

Решение.

```
CREATE SECURITY RULE RULE_2
/* Создать правило безопасности RULE_2*/
      CRANT RETRIEVE (S#, SURNAME, CITY)
      /* Позволить извлечение столбцов (S#, SURNAME,
       CITY) */
      ON _TABLE
      /* из отношения _TABLE */
      TO ANDREW, IRINA, SERGE;
      /* Для пользователей: Андрей, Ирина, Сергей */
```

Пример. Создать правило безопасности для пользователей ANDREW, IRINA, SERGE, позволяющее извлечение, вставку и удаление столбцов S# (номер), SURNAME (фамилия), CITY (город) из отношения _TABLE, если поставщики обладают статусом 5 и находятся в Москве.

Решение.

```
CREATE SECURITY RULE RULE_3
/* Создать правило безопасности RULE_3*/
      CRANT RETRIEVE (S#, SURNAME), INSERT, DELETE
      /* Позволить извлечение столбцов (S#, SURNAME),
       вставку и удаление */
      ON _TABLE WHERE _TABLE.STATUS > 5
      AND _TABLE.CITY = 'Moscow'
      /* из отношения _TABLE, когда у поставщика статус
       более 5 и поставщик находится в Москве */
      TO ANDREW, IRINA, SERGE;
      /* Для пользователей: Андрей, Ирина, Сергей */
```

Контрольный след

Контрольный след выполняемых операций необходим для регистрации выполняемых операций с данными, особенно с критическими операциями. Для сохранения контрольного следа обычно используют *особый файл*, в котором система автоматически записывает все выполненные пользователями операции при работе с базой данных. (В некоторых системах контрольный след может физически сохраняться в файле восстановления данных, а в некоторых — в отдельном файле. В любом случае пользователи должны иметь возможность обращаться к контрольному следу с помощью традиционного языка реляционных запросов.) Типичная запись такого файла может содержать следующую информацию:

- запрос (исходный текст запроса);
- терминал, с которого была вызвана операция;
- пользователь, задавший операцию;

- дата и время запуска операции;
- вовлеченные в процесс исполнения операции базовые отношения, кортежи и атрибуты;
- старые значения;
- новые значения.

Обязательное управление доступом

Методы обязательного доступа применяются в организациях, имеющих достаточно статичную и жесткую структуру (например, правительственные организации или военные). *Основная идея* — каждый объект имеет некоторый *уровень классификации* (например, "секретно", "совершенно секретно", "для служебного пользования" и т. д.), а каждый пользователь имеет *уровень допуска* с такими же градациями, что и в уровне классификации. Уровни должны образовывать строгий иерархический порядок (что идет выше чего). Отсюда вытекают правила безопасности:

- пользователь *i* имеет доступ к объекту *j*, только если его уровень допуска больше или равен уровню классификации объекта *j*;
- пользователь *i* может модифицировать объект *j*, только если его уровень допуска равен уровню классификации объекта *j*.

Поддержка мер обеспечения безопасности в языке SQL

В языке SQL предусматривается поддержка только избирательного управления доступом, которая основана на двух более или менее независимых частях. *Механизм представлений* используется для скрытия очень важных данных от несанкционированных пользователей. *Подсистема полномочий* наделяет пользователей избирательными правами по отношению к объектам БД и действиям с данными.

В SQL правила безопасности задаются на основе директивы GRANT (рис. 5.7), а не с помощью CREATE RULE.

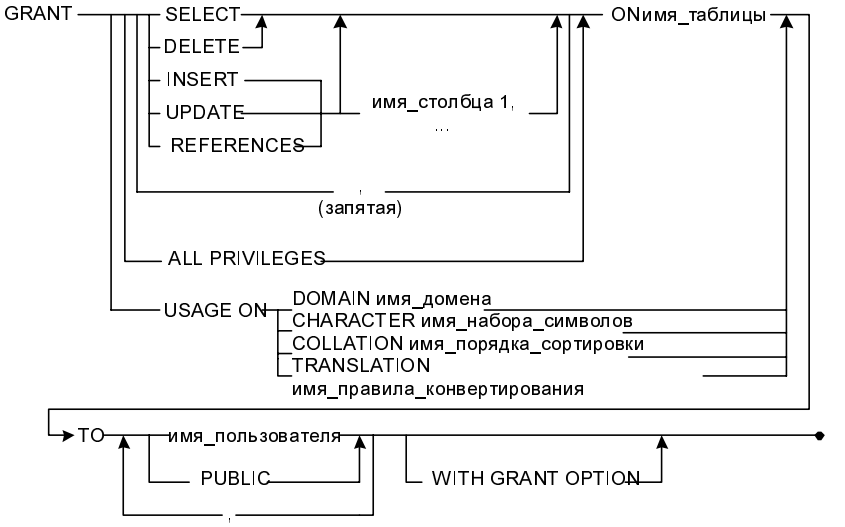


Рис. 5.7. Инструкция предоставления привилегий (GRANT)

При создании таблицы с помощью инструкции CREATE TABLE пользователь автоматически получает для нее все привилегии. Для доступа других пользователей к таблице им следует предоставить привилегии с помощью инструкции GRANT.

В упрощенном виде синтаксис утверждения GRANT (предоставление полномочий) можно записать следующим образом:

```
GRANT список_привилегий_через_запятую
      ON объект
      TO список_пользователей_через_запятую
      [ WITH GRANT OPTION ];
```

Среди допустимых привилегий, т. е. действий, которые пользователь имеет право выполнять над объектом БД, могут

быть: операции использования (USAGE), выбора (SELECT), вставки (INSERT), обновления (UPDATE), удаления (DELETE) и ссылки (REFERENCES). Привилегия USAGE необходима для указания использования некоторого заданного домена, привилегия REFERENCES — для обращения в ограничениях целостности к специально заданной таблице. Привилегии INSERT и UPDATE могут задаваться для специально заданных столбцов.

В качестве объектов могут выступать, например, таблицы и представления.

Список пользователей, разделенных запятыми, может быть заменен ключевым словом PUBLIC, которое означает всех пользователей.

Если задана директива WITH GRANT OPTION, это значит, что указанные пользователи наделены особыми полномочиями для заданного объекта — *правом предоставления полномочий*. Если необходимо дать какие-либо одни привилегии с правом предоставления, а другие — без, то необходимо воспользоваться двумя отдельными инструкциями GRANT — GRANT с директивой WITH GRANT OPTION и без.

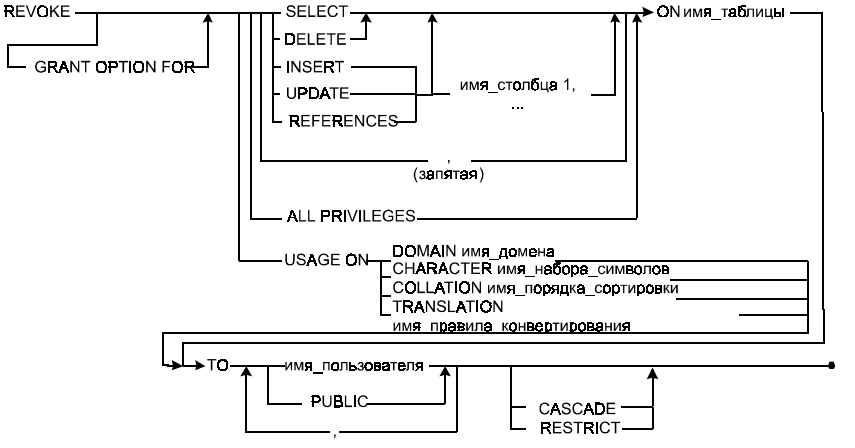


Рис. 5.8. Инструкция отмены привилегий (REVOKE)

Отмена привилегий производится с помощью инструкции REVOKE (рис. 5.8).

Сокращенный вид инструкции REVOKE:

```
REVOKE [ GRANT OPTION FOR ] список_привилегий_через_запятую
      ON объект
      FROM список_пользователей_через_запятую параметр;
```

Директива GRANT OPTION FOR означает, что отменяются только полномочия, назначаемые (передаваемые) другим пользователям; список привилегий и список пользователей используются так же, как и в утверждении GRANT, а в качестве параметра используется либо RESTRICT, либо CASCADE. RESTRICT и CASCADE предотвращают ситуации, появляющиеся с возникновением покинутых привилегий. При задании RESTRICT не разрешается выполнять операцию отмены привилегии, если она приводит к появлению покинутой привилегии. Параметр CASCADE указывает на последовательную отмену всех привилегий, производных от данной.

При удалении домена, таблицы, столбца или представления автоматически будут удалены также и все привилегии в отношении этих объектов со стороны всех пользователей.

Пример. Создать правило безопасности, позволяющее извлечение и обновление столбцов SURNAME, NAME, _YEAR (курс), _GROUP, FACULTY таблицы STUDENTS, если студенты обучаются с 1 по 6 курс на историческом, филологическом или экономическом факультетах и проживают в городах Гродно или Минске, для пользователей Deanery_1, Deanery_2.

Решение.

```
CREATE VIEW STUDIES AS
       SELECT SURNAME, NAME, _YEAR, _GROUP, FACULTY
       FROM STUDENTS
       WHERE FACULTY IN ('исторический',
       'филологический', 'экономический')
       AND _YEAR IN (1, 2, 3, 4, 5, 6)
```

```
AND CITY = 'Гродно' OR CITY = 'Минск';
GRANT SELECT, INSERT, DELETE ON STUDIES TO
Deanery_1, Deanery_2;
```

Пример. Создать правило безопасности для пользователей `Deanery_3`, `Deanery_4`, позволяющее извлечение, удаление и обновление столбцов `COURSE_NAME`, `TEACHER`, `_HOURS` таблицы `COURSE`, если курсы включены в 1, 3 или 5 семестры, и количество их часов находится в пределах от 30 до 90.

Решение.

```
CREATE VIEW STUDR AS
       SELECT SUBJECT_NAME, LECTURER, _HOURS
       FROM SUBJECTS
       WHERE SEMESTER IN (1, 3, 5)
       AND _HOURS >= 30 AND _HOURS <= 90;
       GRANT SELECT, INSERT, DELETE ON STUDR TO
       Deanery_3, Deanery_4;
```

Задания

1. В чем заключается корректность базы данных?
2. Как можно классифицировать ограничения целостности?
3. Для чего используются в базе данных триггеры?
4. Определить в общем виде синтаксис для создания триггера.
5. С какой целью создаются в базе данных генераторы?
6. Как происходит обращение к генератору?
7. В чем преимущества использования хранимых процедур?
8. Определить в общем виде синтаксис для создания хранимой процедуры.
9. Привести примеры операторов, которые используются в теле хранимых процедур и триггеров.

10. Для чего применяются функции в базе данных?
11. Что такое восстановление системы? Какие виды нарушений системы существуют?
12. Перечислить ACID-свойства транзакций.
13. Чем обеспечивается в базах данных управление транзакциями?
14. Что такое точка отката?
15. Для чего в СУБД используется журнал транзакций?
16. Что такое прямое и обратное восстановление базы данных?
17. В чем заключается управление параллельным доступом?
18. Объяснить на примерах концепцию сериализации транзакций.
19. Какие проблемы возникают при параллельной обработке данных? Привести соответствующие примеры из предметной области.
20. Что такое блокировка?
21. Какие уровни блокировки могут быть реализованы в СУБД?
22. Когда используется жесткая и нежесткая блокировки?
23. Когда в СУБД может возникнуть тупиковая ситуация?
24. Что такое уровень изоляции?
25. В каких ситуациях следует использовать следующие уровни изоляции: `SERIALIZABLE`, `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED`?
26. В чем состоит упорядоченность транзакций?
27. Что такое безопасность базы данных?
28. Какую можно дать классификацию нарушениям при работе с базой данных?
29. Какие можно выделить компьютерные средства контроля для защиты базы данных?

30. Какие методы включаются в некомпьютерные средства контроля?
31. В чем состоит избирательный подход к обеспечению безопасности данных?
32. В чем состоит обязательный подход к обеспечению безопасности данных?
33. Что такое правило безопасности?
34. С помощью какой инструкции в SQL задаются правила безопасности?
35. Создать утверждение, которое проверяет, что плановый объем продаж каждого офиса не будет превышать суммарные плановые объемы продаж служащих данного офиса (см. рис. 4.5).
36. Создать утверждение, которое проверяет, что плановый объем продаж каждого офиса будет совпадать с суммарными плановыми объемами продаж служащих данного офиса (см. рис. 4.5).
37. Создать утверждение, которое для каждого студента, обучающегося на третьем, четвертом либо пятом курсе, со средним баллом успеваемости выше четырех устанавливает 25% надбавку к стипендии (см. рис. 4.3 с поправкой к таблице `STUDENTS`: добавлено поле — `STIPEND`).
38. Создать утверждение, устанавливающее всем студентам факультетов, у которых средняя оценка факультета за летнюю сессию выше общей средней оценки по базе, 10% надбавку к стипендии (см. рис. 4.3 с поправкой к таблице `STUDENTS`: добавлено поле — `STIPEND`).
39. Создать утверждение, устанавливающее студентам математического, физического и экономических факультетов пятого курса, у которых средний балл больше либо равен 4,5, 50% надбавку к стипендии (см. рис. 4.3 с поправкой к таблице `STUDENTS`: добавлено поле — `STIPEND`).
40. Создать триггер, реализующий каскадное изменение в таблице `SESSION_RESULTS` при изменении личных данных (в частности, фамилии) в таблице `STUDENTS` (см. рис. 4.3).

41. Создать генератор и соответствующий триггер, обеспечивающий уникальные значения для столбца STUDENT_ID таблицы STUDENTS (см. рис. 4.3).

42. Создать триггер, реализующий каскадное изменение в таблице EMPLOYEE при изменении соответствующих данных, касающихся номера офиса в таблице OFFICES (см. рис. 4.5).

43. Создать генератор и соответствующий триггер, обеспечивающие уникальные значения для столбца ORDER_ID таблицы ORDERS (см. рис. 4.5).

44. Создать триггер, реализующий каскадное изменение в таблице ORDERS при изменении соответствующих данных, касающихся работников в таблице EMPLOYEE (см. рис. 4.5).

45. Создать генератор и соответствующий триггер, обеспечивающий уникальные значения для столбца CLIENT_ID таблицы CLIENTS (см. рис. 4.5).

46. Создать хранимую процедуру, выводящую общее количество студентов, обучающихся в вузе, и среднюю оценку студента (рис. 4.3).

47. Создать хранимую процедуру, выводящую средний бал успеваемости по каждому факультету и общее количество студентов факультета (рис. 4.3).

48. Создать хранимую процедуру, выводящую общее количество часов для каждого преподавателя (рис. 4.3).

49. Создать хранимую процедуру, выводящую общую стоимость и общее число заказов клиентов (рис. 4.5).

50. Создать хранимую процедуру, выводящую общую стоимость и общее число заказов клиентов по каждому офису (рис. 4.5).

51. Создать хранимую процедуру, выводящую общее количество заказов, каждое из которых превышает среднюю стоимость заказа по всей базе данных (рис. 4.5).

52. Создать правило безопасности, позволяющее извлечение, вставку и удаление для всех столбцов таблицы STUDENT,

если студенты обучаются с 4 по 6 курс на математическом, историческом или экономическом факультетах, для пользователей `Deanery_1`, `Deanery_2` (рис. 4.3).

53. Создать правило безопасности, позволяющее извлечение, вставку и удаление для всех столбцов таблицы `STUDENT`, если студенты обучаются с 1 по 3 курс на биологическом, филологическом или юридическом факультетах, для пользователей `Deanery_3`, `Deanery_4` (рис. 4.3).

54. Создать правило безопасности, позволяющее извлечение, удаление и обновление столбцов `SUBJECT_NAME`, `LECTURER`, `APPOINTMENT`, `_HOURS` таблицы `SUBJECT`, если курсы включены во 2, 4 или 6 семестры, и их количество часов находится в пределах от 50 до 110 для пользователей `Deanery_5`, `Deanery_6` (рис. 4.3).

55. Создать правило безопасности, позволяющее извлечение и обновление столбцов `SURNAME`, `NAME`, `_YEAR`, `_GROUP`, таблицы `STUDENT`, если студенты обучаются с 3 по 6 курс на математическом, филологическом или экономическом факультетах и проживают в городах Гродно, Минск, Москва или Псков, для пользователей `Deanery_7` и `Deanery_8` (рис. 4.3).

Глава 6

Создание приложений средствами Microsoft Access

MS Access представляет собой приложение Microsoft Office, которое позволяет создавать программные средства для получения определенных результатов. Главное отличие MS Access от других СУБД заключается в том, что под базой данных MS Access понимается совокупность структурированных и взаимосвязанных данных и методов, обеспечивающих добавление, изменение, выборку и отображение данных. Как правило, многие системы, связанные с БД, позволяют хранить все таблицы одном файле и не включают формы и отчеты в файл БД. Для добавления, изменения и отображения данных в таких системах используют приложения конечного пользователя, которые называются также терминальными приложениями.

Общие замечания по созданию баз данных средствами Microsoft Access

Основу построения MS Access составляют реляционные БД. В силу этого необходимы определенные сведения по структу-

ре, организации, функциям БД, а также процессу проектирования. В конечном итоге именно процесс проектирования и приводит к созданию оптимальной в некотором смысле БД, являющейся основой для создания программного средства (приложения), которое позволяет получать необходимые результаты из совокупности хранимых данных.

Одно из главнейших требований при работе с БД — разработка ее проекта. В настоящее время методология концептуального проектирования является ведущей при создании логических проектов БД и предполагает наличие определенной квалификации и опыта для анализа предметной области и построения соответствующего проекта будущей системы со всеми необходимыми требованиями.

Работа с Microsoft Access предполагает создание определенных объектов БД — таблиц, запросов, форм, отчетов, макросов, модулей. Однако в зависимости от требований предметной области не все объекты СУРБД MS Access могут быть использованы при создании конкретного приложения. В частности, речь идет о макросах, модулях и страницах данных. Использование этих объектов предполагает достаточно высокие требования к конечному программному продукту.

При создании проекта СУРБД MS Access можно придерживаться следующих этапов:

1. Разработка общих концепций будущей системы — выяснение требований пользователей и сбор информации по конкретной предметной области: зачем нужна эта система, какие ее данные будут востребованы и т. д.
2. Проектирование отчетов, которое предполагает планирование состава и внешнего вида выходных документов создаваемого приложения.
3. Определение структуры данных, которая является основой создаваемого приложения. Данные, полученные на первых двух этапах, анализируются и разбиваются по группам, исключающим повторение, но предусматривающим определенную взаимосвязь и восприятие их как единого целого в

виде совокупности таблиц с необходимыми заголовками. Связь между таблицами осуществляется в виде повторяющихся полей, т. е. в одной таблице данные являются уникальными (не имеют повторений в пределах данного столбца), а в другой — возможно дублирование данных в аналогичном столбце. На этом этапе предусматривается также детальная проработка полей таблиц, типов хранимых данных, связей между таблицами, ограничений на данные и т. д.

4. Разработка структуры запросов к БД, т. е. определение тех выборок данных из базы, которые необходимы для создания выходных документов (отчетов) и получения определенной информации, участвующей в тех или иных визуальных либо печатных документах приложения.

5. Проектирование экранных форм приложения. Этот этап можно разделить на два:

   - создание пользовательских окон, предназначенных для ввода и редактирования информации (в таблицы);

   - создание форм, выводящих результаты обработки данных, которые будут представлены либо в определенном окне создаваемого приложения, либо в виде готовых отчетов, формируемых приложением (см. п. 2).

6. Придание приложению MS Access законченного вида. На этом этапе разрабатываются кнопочные и обычные меню, а также ключевые формы приложения для облегчения работы пользователей СУРБД.

Следует заметить, что на каждом этапе проектирования следует тщательно анализировать результаты предыдущего этапа и, при необходимости, вносить коррективы в требования по разработке тех или иных объектов будущего приложения.

После того как процедура проектирования выявила все необходимые требования, можно приступать к реализации приложения средствами СУРБД MS Access.

Особенности интерфейса Microsoft Access

Приложение Microsoft Access достаточно специализировано и предназначено, в первую очередь, для разработчиков БД. Основным компонентом проекта MS Access является БД, включающая в свой состав таблицы, запросы, формы, отчеты, страницы, макросы и модули.

Одна из особенностей работы с MS Access заключается в том, что нельзя изначально открыть новый файл — файл, в котором будут отображены основные компоненты приложения для создания БД, а также сохранить при последующей работе весь файл БД. Для создания новой БД необходимо указать ее местоположение, выбрав сначала команду **Создать** в меню **Файл**, а затем на панели инструментов **Область задач** из категории **Создание** команду **Новая база данных**. Соответственно, для открытия существующей БД необходимо использовать команду **Открыть** в меню **Файл**. Сохранение всех объектов БД будет фиксироваться в файле с тем именем, который задан первоначально, и выполнено в формате Microsoft Access 2000. В дальнейшем в MS Access можно сохранять лишь отдельные объекты БД (для объекта доступна команда **Сохранить как** в меню **Файл** либо соответствующая команда контекстного меню с допустимыми параметрами). Для завершения работы с БД достаточно выполнить команду **Закрыть** в меню **Файл** (без закрытия самого приложения MS Access), либо команду **Выход** в меню **Файл** (с завершением работы приложения), либо стандартную кнопку закрытия  в окне БД.

Окно приложения MS Access с открытой БД, которая представлена в своем окне — *окне БД* — является типовым окном Windows, а также аналогично окнам других приложений пакета Microsoft Office (рис. 6.1).

Кнопка системного меню приложения MS Access имеет вид  , а кнопка системного меню окна БД —  .

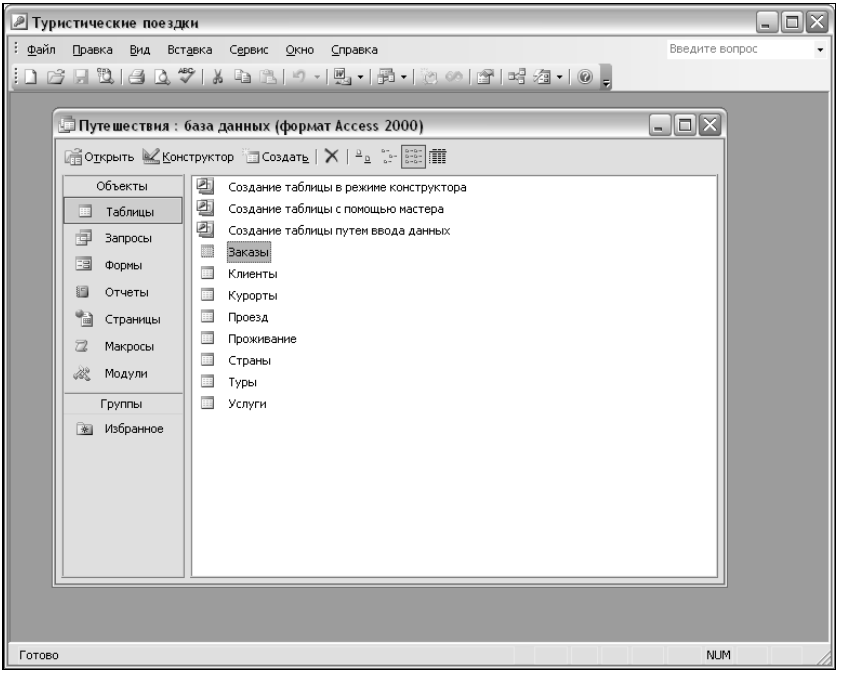


Рис. 6.1. Главное окно MS Access с открытой БД

Главное окно MS Access предназначено для работы с БД. Из него можно вызвать любой объект БД для просмотра, выполнения, разработки или модификации.

Для работы с конкретной БД можно разработать собственный интерфейс, основой которого, как правило, являются формы. На формах располагаются различные элементы: поля таблиц, поля со списком, кнопки, раскрывающиеся списки, выключатели, переключатели, флажки, рисунки, подчиненные формы и т. д. За кнопками обычно закрепляют вызов функций. Все функции обработки информации задаются с помощью макросов или программ на VBA. В приложениях для работы с БД обычно помещают главную кнопочную форму (рис. 6.2), которая автоматически появляется при открытии БД. В дальнейшем работа пользователя происходит

не только с помощью главной формы, но и других форм и окон. Базу данных, имеющую интерфейс пользователя и расширение mde, можно считать приложением, т. к. все описания БД, включая программные коды, интерпретируются системой MS Access при работе пользователя.

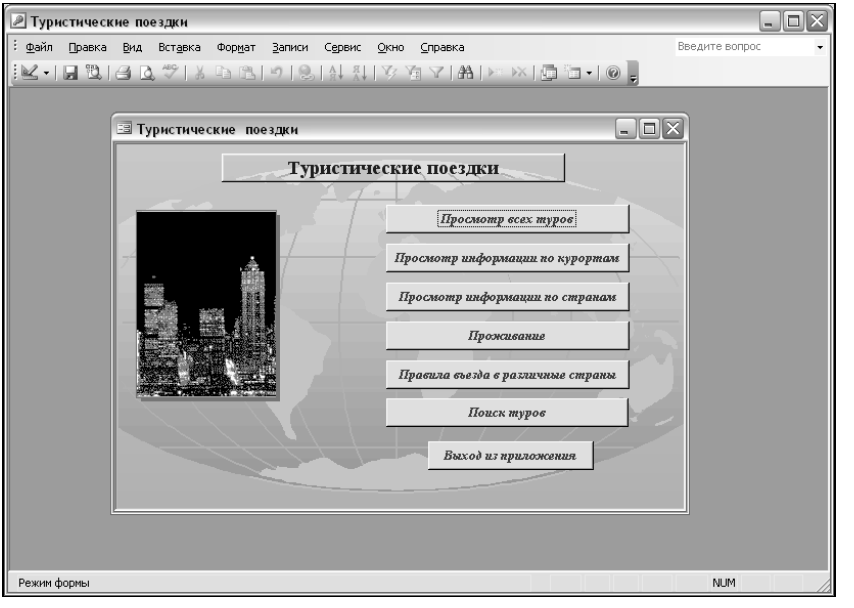


Рис. 6.2. Главная кнопочная форма БД "Путешествия"

Работающая БД MS Access может взаимодействовать с внешними БД, которые могут использоваться как источник таблиц при импорте или присоединении, а также как получатель при экспорте данных из работающей БД. В качестве внешней БД может служить любая БД, поддерживающая протокол ODBC (Open DataBase Connectivity, набор интерфейсов прикладного уровня), например, база данных SQL Server, расположенная на удаленном сервере.

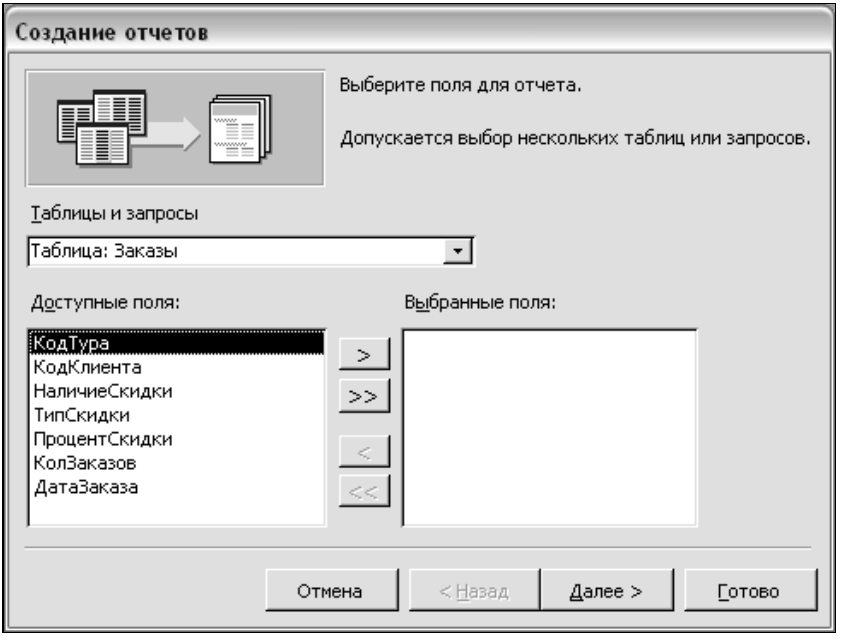


Рис. 6.3. Создание отчета с помощью мастера отчетов

Основу СУРБД Microsoft Access составляют 7 объектов. Эти объекты включают данные и различные инструментальные средства, необходимые для использования MS Access. Многие объекты можно создавать как с использованием мастера (рис. 6.3), так и с использованием конструктора (рис. 6.4).

Таблица — основная единица хранения данных в базе. Понятие таблицы в MS Access соответствует понятию двумерной таблицы (запись × поле) реляционной модели данных. В любой базе, как правило, имеется некоторое число связанных между собой таблиц. Между двумя таблицами устанавливаются связи типа "один-к-одному" и "один-ко-многим" командой **Схема данных** в меню **Сервис** (рис. 6.5). Среди основных операций над таблицами можно выделить: просмотр и обновление (ввод, модификация и удаление), сортировку, фильтрацию и печать.

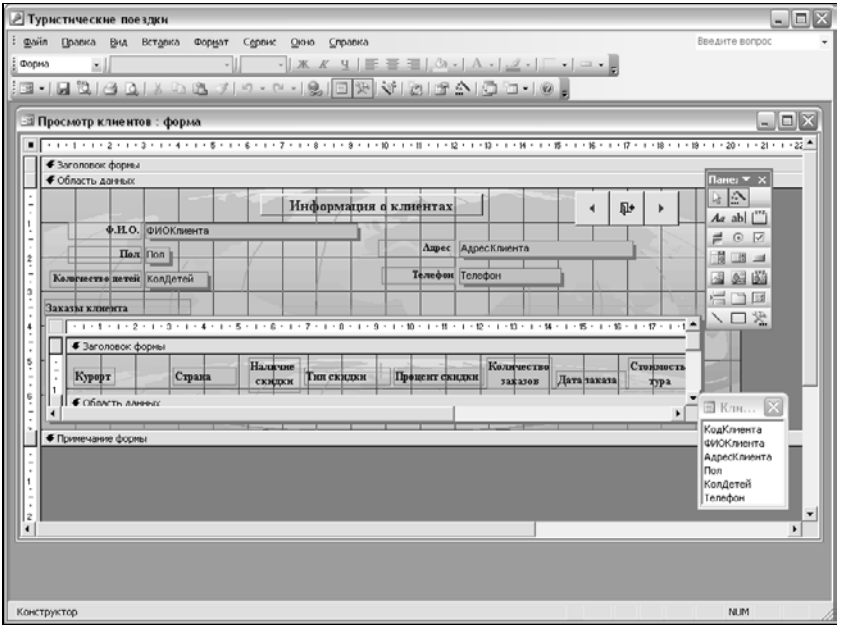


Рис. 6.4. Создание формы в режиме конструктора форм

Запросы позволяют осуществить выборку данных по некоторому критерию или выполнить определенные действия с данными. Одновременно выборка может производиться из 16 таблиц. В запрос можно включать до 255 полей. В MS Access можно создавать и выполнять следующие основные типы запросов: *на выборку* — из одной и многих таблиц; *перекрестный запрос*; *модифицирующие запросы* (обновление, удаление или добавление данных). С помощью запросов можно создавать новые таблицы, используя данные из одной или нескольких существующих таблиц. Описание запроса можно выполнить с помощью бланка QBE (Query-by-Example, язык запросов по образцу) или SQL.

Форма представляет собой объект БД MS Access, в котором можно разместить элементы управления, принимающие дей-

ствия пользователей или служащие для ввода, отображения и изменения данных в таблицах или запросах.

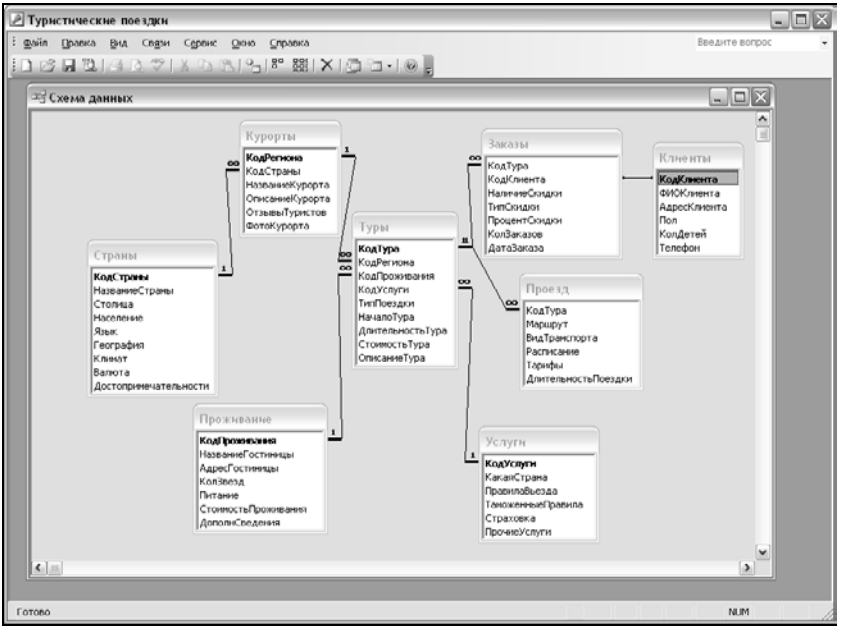


Рис. 6.5. Окно **Схема данных**

Отчеты предназначены для печати данных, содержащихся в таблицах и запросах в соответствии с некоторыми требованиями оформления.

Страница доступа к данным (рис. 6.6) — это страница, которую можно использовать для добавления, редактирования, просмотра или манипулирования текущими данными в БД MS Access или SQL-сервера. Можно создать страницы для ввода и редактирования данных аналогично формам MS Access, или вывода иерархически сгруппированных записей подобно отчетам MS Access.

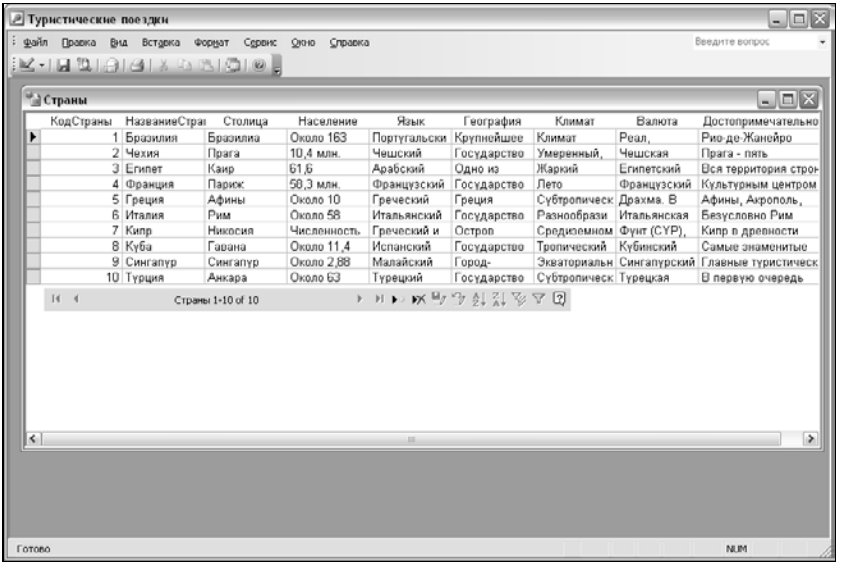


Рис. 6.6. Страница доступа к данным

Страницы доступа к данным предлагают определенные возможности.

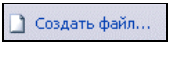
- Сбор и распределение текущих данных несколькими способами, т. к. их можно использовать для добавления, редактирования и просмотра данных в БД или проекте MS Access. Страницы доступа к данным можно использовать в Интернете или интранете (intranet) и посылать по электронной почте.
- Интерактивный просмотр сгруппированных записей. На страницах с группами записей можно отображать нужные детали путем раскрытия или удаления заголовков групп. Имеются средства сортировки и фильтрации записей.
- Анализ данных и проектирование. На странице можно получить результаты выполнения перекрестного запроса, произвести вычисления, используя MS Excel, просмотреть данные с помощью диаграммы.

- Выдача текста страницы в формате HTML. Можно сохранить HTML-код в поле своей БД и вывести его как форматированный текст HTML на странице.
- Привычная среда разработки, т. к. в режиме конструктора страниц используются различные панели инструментов и другие возможности, применяемые при создании форм и отчетов.

Макросы представляют собой последовательность макрокоманд встроенного языка MS Access, которые задают автоматическое выполнение определенных операций.

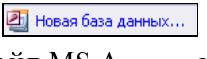
Модуль — это совокупность описаний, инструкций и процедур на языке VBA, сохраненная под общим именем. В MS Access используются модули трех типов: формы, отчета и стандартный. Модули форм и отчетов содержат программы, являющиеся локальными для этих объектов. Процедуры из стандартного модуля, если они не описаны явно как локальные для содержащего их модуля, распознаются и могут вызываться процедурами из других модулей в той же БД или из администрируемых БД.

Создание базы данных

В Microsoft Access имеются несколько возможностей по созданию БД. Команда **Создать** в меню **Файл** либо кнопка 

в области задач выводят список всех имеющихся возможностей по созданию базы данных, показанных на рис. 6.7.

Создание новой базы данных

- Для создания новой БД необходимо воспользоваться кнопкой 

в области задач. При этом формируется файл MS Access, куда будут помещаться все необходимые объекты.

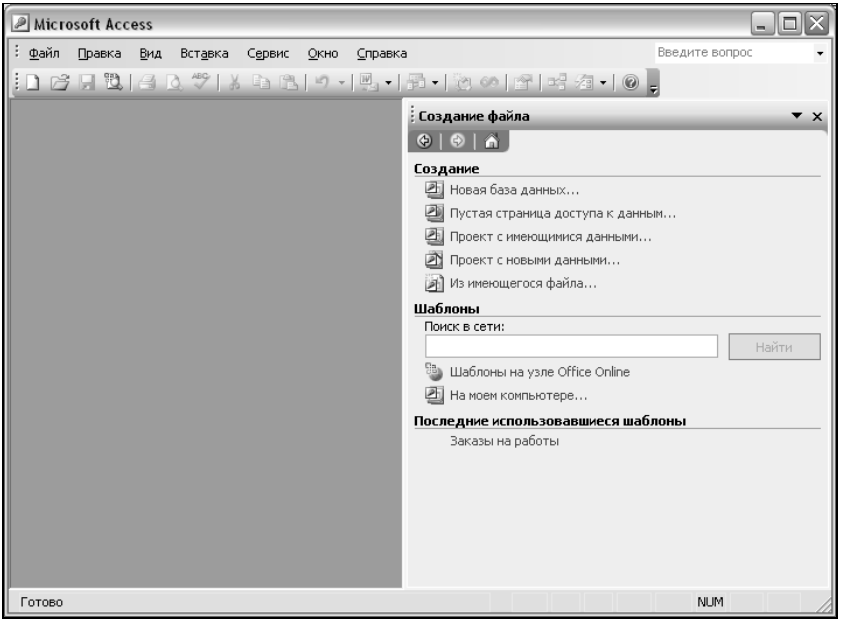


Рис. 6.7. Возможности создания файла БД в Microsoft Access

- При создании пустой страницы доступа к данным (кнопка 

в области задач) создается Web-форма, связанная с MS Access или SQL Server, которую можно разместить в корпоративной интранет-сети.
- Проект с имеющимися данными (кнопка 

в области задач) позволяет с помощью Microsoft Database Engine (диалоговое окно **Свойства связи с данными**, рис. 6.8) создать новый проект SQL Server (ADP-файл) для хранения таблиц данных. Имеющийся файл MS Access используется как клиентское приложение (приложение конечного пользователя).
- При нажатии кнопки 

создается как новая БД MS Access (*.mdb), так и новый проект SQL Server (*.adp) для хранения таблиц данных.

- Кнопка 

позволяет создать БД из готовых файлов, которые имеются на компьютере.

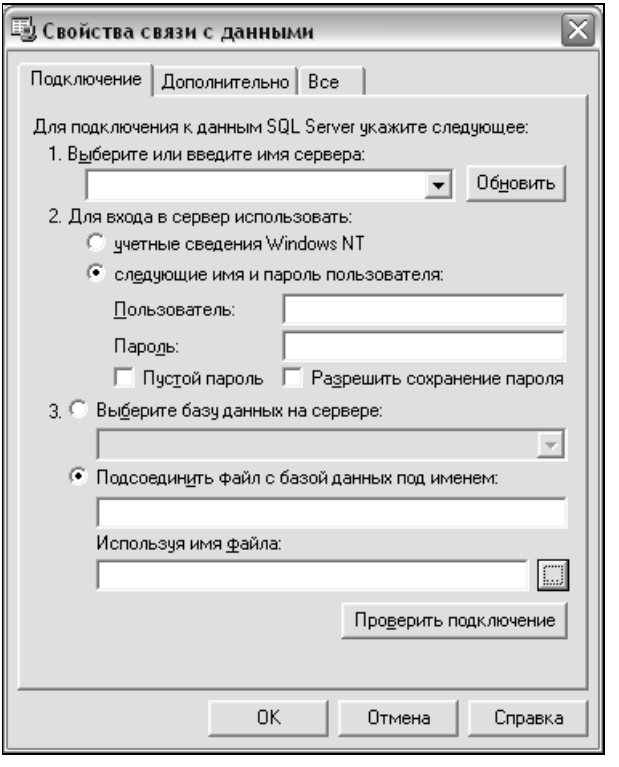
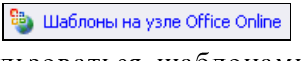


Рис. 6.8. Диалоговое окно **Свойства связи с данными**

Создание базы данных на основе шаблонов

Так же как и при создании новой БД, в MS Access предусмотрено несколько способов создания БД на основе шаблонов.

- Кнопка 

в области задач позволяет воспользоваться шаблонами БД, находящимися в Интернете.

- Доступ к имеющимся БД на рабочем компьютере осуществляется кнопкой На моем компьютере..., открывающееся при этом окно **Шаблоны**, в частности, вкладка **Базы данных** (рис. 6.9), позволяет создать либо использовать БД из готовых проектов.

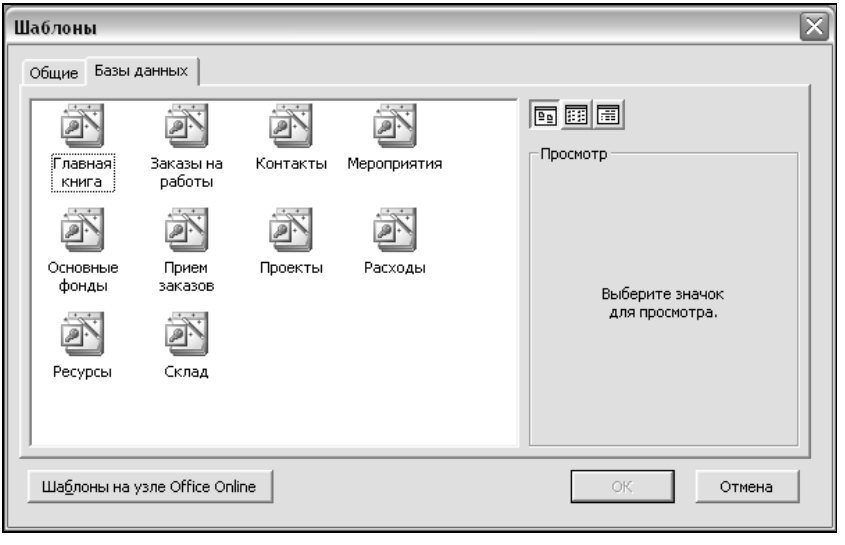


Рис. 6.9. Создание БД из шаблона

При создании файла новой БД (сразу же после запуска процесса создания) необходимо указать имя и выбрать местоположение будущей БД. В созданном файле рабочим окном по созданию и модифицированию соответствующих объектов является окно БД (рис. 6.10), в котором для редактирования выбирается один из объектов БД (таблицы, запросы и др.). Следует отметить, что созданная новая БД без использования шаблонов либо имеющихся файлов первоначально не будет содержать никаких объектов.

Для создания и редактирования какого-либо объекта БД необходимо перейти на соответствующую вкладку окна БД, связанной с данным объектом. Панель инструментов окна БД

(рис. 6.11) предоставляет необходимые возможности при работе с объектами БД (табл. 6.1).

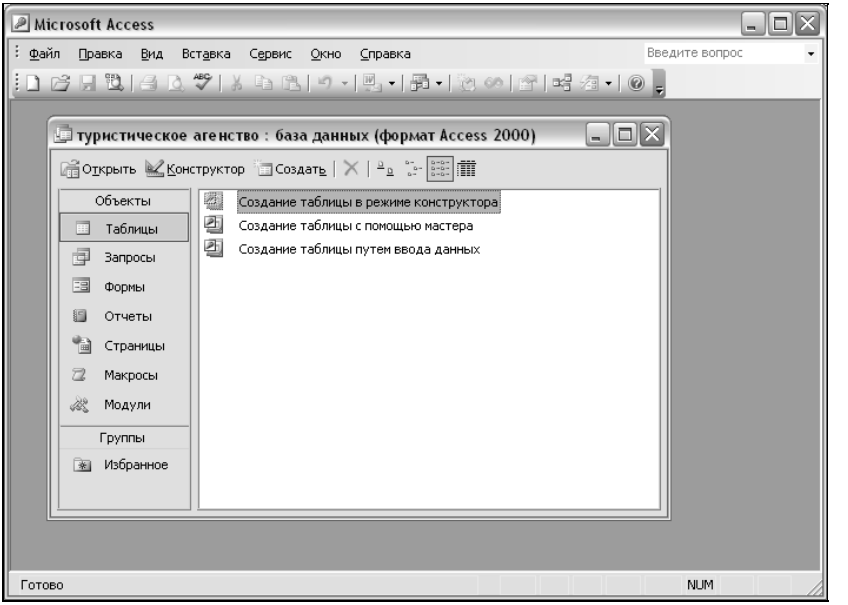


Рис. 6.10. Окно пустой БД

Рис. 6.11. Панель инструментов окна БД

Таблица 6.1. *Возможности кнопок панели инструментов окна БД при работе с объектами*

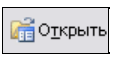
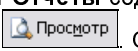
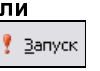
Изображение кнопки	**Назначение**	**Примечание**
	Открывает выделенный объект БД для просмотра	Объект **Отчеты** содержит кнопку  . Объекты **Макросы** и **Модули** содержат кнопку 

Таблица 6.1 *(продолжение)*

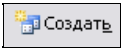
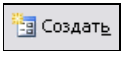
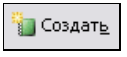
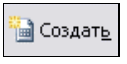
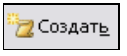
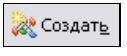
Изображение кнопки	Назначение	Примечание
	Открывает выделенный объект в режиме редактирования	Объект **Модули** содержит следующую кнопку: 
	Удаляет выделенный объект БД	Не всегда любой объект БД можно удалить
Кнопки для создания объектов БД		
	Создание объекта **Таблица**	Кнопки предлагают различные возможности по созданию конкретных объектов БД. Кроме того, возможности создания объектов вынесены также в содержимое окна конкретного объекта (см. рис. 6.14)
	Создание объекта **Запрос**	
	Создание объекта **Форма**	
	Создание объекта **Отчет**	
	Создание объекта **Страница**	
	Создание объекта **Макрос**	
	Создание объекта **Модуль**	
Кнопки изменения представления объектов в окне БД		
	Представляет объекты окна в виде крупных значков	Кнопки изменения вида объектов, находящихся в БД. Позволяют изменять визуальное представление объектов в окне БД

Таблица 6.1 *(окончание)*

Изображение кнопки	Назначение	Примечание
Кнопки изменения представления объектов в окне БД		
	Представляет объекты окна в виде мелких значков	
	Представляет объекты окна в виде списка	
	Представляет объекты окна в виде таблицы с указанием некоторых дополнительных данных	

Создание таблиц и схемы данных

Основными объектами БД являются таблицы, которые связаны между собой в схему данных. Именно с этих объектов необходимо начинать работу.

Общие рекомендации по созданию таблиц и схемы данных

В MS Access существуют различные возможности по созданию таблиц, описанные в табл. 6.2.

Таблица 6.2. *Возможности создания таблиц в MS Access*

Режим	Описание
Режим таблицы	Необходимый макет таблицы формируется соответствующим вводом полей в заголовок таблицы, добавлением либо удалением соответствующих столбцов (для этого удобно использовать контекстное меню выделенного столбца) и т. д.

Таблица 6.2 *(окончание)*

Режим	Описание
	Не закрывая подготовленного макета таблицы, вводятся данные в строки таблицы. При сохранении таблицы автоматически анализируются данные и соответствующему полю присваивается необходимый тип данных; кроме того, происходит запрос о ключевом поле и об имени таблицы
Конструктор	Формирование структуры таблицы происходит при заполнении соответствующего бланка, в котором указываются необходимые поля, тип данных, свойства поля и т. д. Режим конструктора предоставляет наиболее широкие возможности по созданию объектов таблицы, и его удобно использовать для внесения необходимых корректив в таблицы, созданные различными способами
Мастер таблиц	Новая таблица формируется на основе существующих таблиц, имеющих различные поля. Наличие понятных рекомендаций при выборе режима мастера таблиц не вызывает сложностей при создании новой таблицы
Импорт таблиц	Осуществляется импорт данных и объектов из внешнего файла в текущую БД
Связь с таблицами	Создаются таблицы, связанные с таблицами внешнего файла

Работа по созданию таблиц и схемы данных может быть разбита на следующие этапы:

1. Создание и определение структуры таблиц:
   - создание новой таблицы;
   - определение полей, типов данных, описаний (при необходимости) и свойств полей, включая маски ввода и условия на значение;
   - задание первичного ключа (возможны составные);
   - создание индексов для необходимых полей;
   - сохранение таблицы в базе.

2. Связывание таблиц в схему данных с учетом требований целостности данных.
3. Определение полей подстановки для удобства работы с данными.

Для создания новой таблицы необходимо перейти к объекту **Таблицы** в левой части окна БД (рис. 6.12) и воспользоваться одной из возможностей по созданию таблиц (табл. 6.2), предоставляемых выбором команды **Таблица** в меню **Вставка** либо кнопкой 

в окне БД.

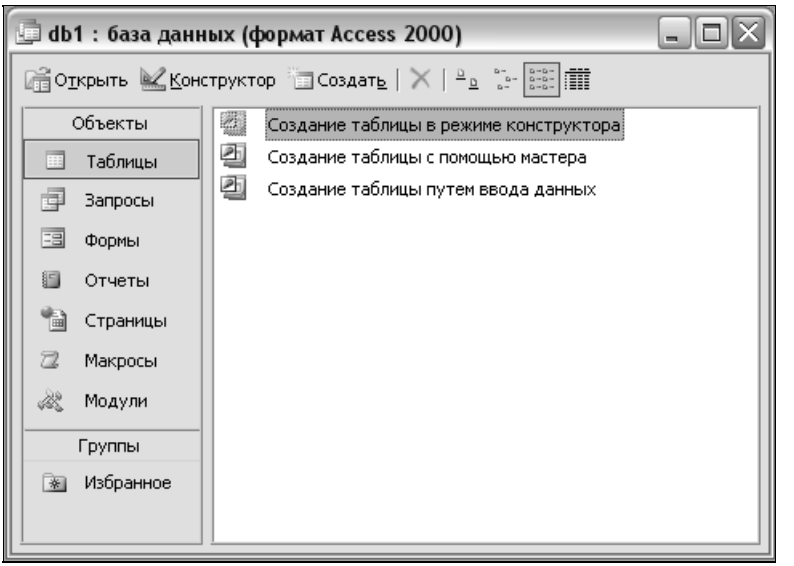


Рис. 6.12. Окно базы данных с активной вкладкой **Таблицы**

Создание таблицы в режиме конструктора

При создании таблицы с использованием режима конструктора в соответствующем окне (рис. 6.13) необходимо в облас-

ти описания полей заполнить столбцы **Имя поля**, **Тип данных**, **Описание**, а также при необходимости — ключевое поле (или поля), а в области **Свойства поля** ввести необходимые свойства для каждого поля с использованием вкладок **Общие** и **Подстановка**.

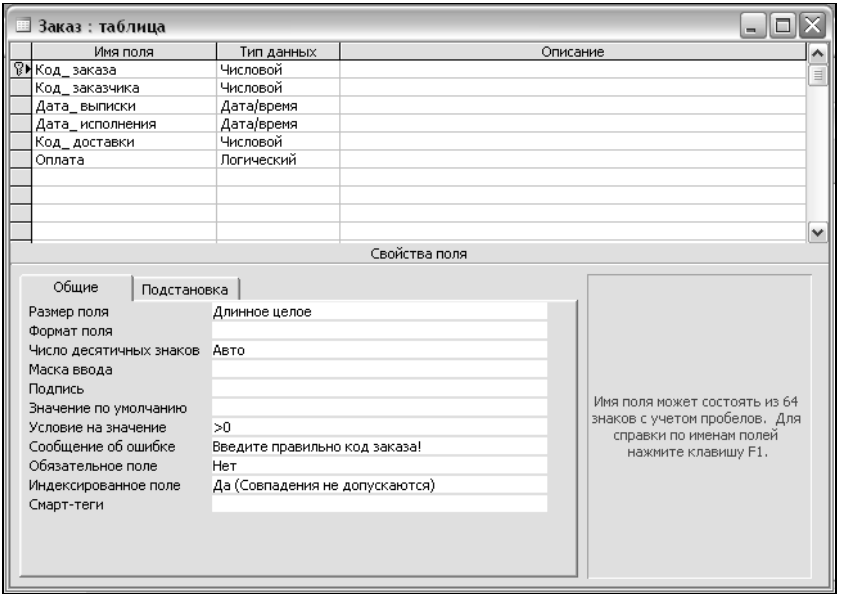


Рис. 6.13. Окно конструктора таблиц

Панель инструментов **Конструктор таблиц** предоставляет широкие возможности для создания структуры новой таблицы (рис. 6.14).

Рис. 6.14. Панель инструментов **Конструктор таблиц**

Кнопка  на панели инструментов Конструктор таблиц позволяет сразу же после создания нового бланка определить параметры таблицы в целом. Так, например:

- *описание* — определение комментария, содержащего описание таблицы в окне БД (чтобы его увидеть, необходимо выделить таблицу и выполнить команду **Таблица** в меню **Вид**);
- *условие на значение* — требование на ввод данных, которое обеспечивает целостность и непротиворечивость данных. Это требование можно применить ко всем полям таблицы;
- *сообщение об ошибке* — текстовое сообщение, которое появляется в случае, если нарушается условие на значение;
- *фильтр* — определение записей, которые появятся после применения фильтра к таблице;
- *порядок сортировки* — задание порядка сортировки записей в таблице и др.

Непосредственно в окне конструктора таблиц (рис. 6.13), как указывалось ранее, задаются следующие параметры.

Имя поля — в данном столбце определяется название поля в таблице, которое должно удовлетворять определенным соглашениям об именах объектов (не более 64 символов), исключающее использование символов ".", "!", "`", "[" и "]". Имя поля не должно начинаться с пробела и содержать управляющие символы (с кодами ASCII 00-31), а также — каждое поле должно быть уникальным в пределах одной таблицы.

Тип данных — столбец предназначен для задания типа данных, которые будут храниться в соответствующем поле. После выбора типа данных из списка становится доступной нижняя часть окна — **Свойства поля** (рис. 6.15).

Список типов данных в MS Access приведен в табл. 6.3.

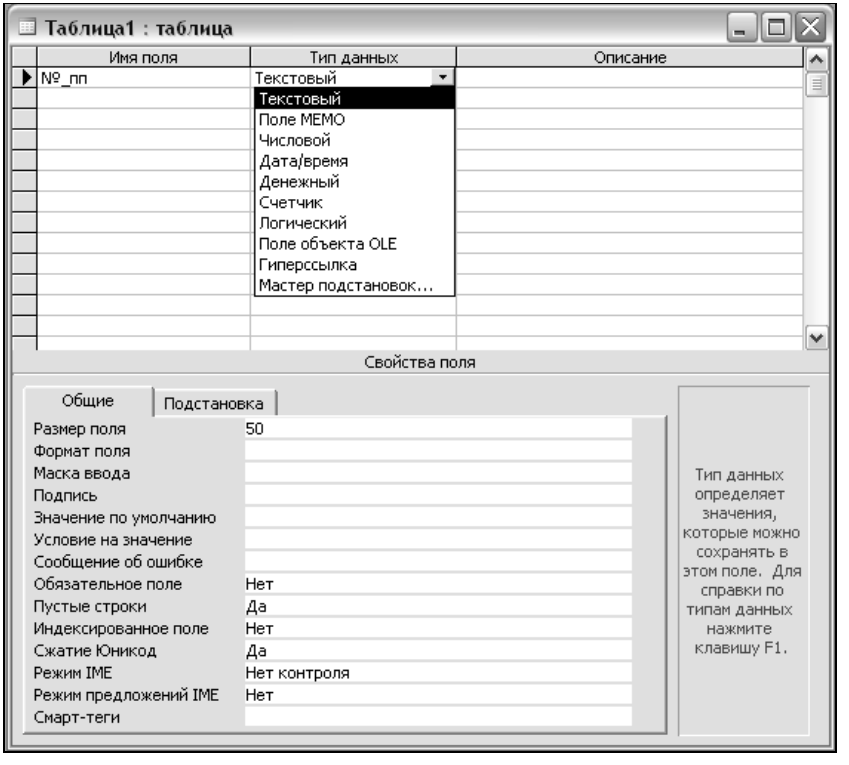


Рис. 6.15. Выбор типа данных для поля таблицы

Таблица 6.3. *Типы данных Microsoft Access*

Тип данных	**Размер**	**Описание**
Текстовый	0—255 символов	Алфавитно-цифровые символы. Применяется для описаний данных текстового характера, а также числовых данных, которые не используются в вычислениях. По умолчанию MS Access ограничивает длину текстовых полей до 50 символов

Таблица 6.3 *(продолжение)*

Тип данных	Размер	Описание
Поле MEMO	0—64 000 символов	Алфавитно-цифровые символы, т. е. текст произвольной длины. Для хранения значений в поле различной длины используется столько места, сколько требуется
Числовой	1, 2, 4 или 8 байт	Числовые значения, используемые в вычислениях. Исключения составляют данные денежного типа, при выполнении вычислений над которыми могут задаваться также и различные типы валют
Дата/Время	8 байт	Данные предназначены для хранения значений дат, времени или обеих этих величин
Денежный	8 байт	Денежные значения, которые обеспечивают точность до 15 знаков слева и 4 знака справа после разделителя целой и дробной частей числа
Счетчик	4 байт	Используется для хранения целых числовых значений, которые автоматически увеличиваются при переходе к новой записи. Можно использовать в качестве уникального идентификатора, т. е. в качестве первичного ключа
Логический	1 бит (0 — значение "Нет", 1 — значение "Да")	Данные этого типа часто используются со специальными элементами управления
Поле объекта OLE	До 1 Гбайт	Данные, обеспечивающие связь с OLE-сервером (рисунки, диаграммы, видео, звук). Объекты такого типа можно просматривать на компьютере в том случае, если имеются необходимые аппаратные возможности и программное обеспечение OLE-сервера

Таблица 6.3 *(окончание)*

Тип данных	Размер	Описание
Гипер-ссылка	До 2048 символов	Адрес ссылки на документ, расположенный в Интернете, локальной сети или на компьютере пользователя
Мастер подстановок	Обычно 4 байт	Отображает данные, подставляемые из другой таблицы. Такие данные удобны для хранения ключевых полей из другой таблицы для связи с данными текущей таблицы

Описание — в данный столбец можно ввести текст описания поля таблицы, который будет отображаться в строке состояния при добавлении или изменении данных в поле.

После ввода имен полей, их типов данных и описаний можно задать более точные свойства для каждого поля таблицы, используя две вкладки **Свойства поля** в нижней части окна конструктора таблиц (рис. 6.15). У каждого поля имеются свойства, которые отличаются у данных различных типов. Краткий перечень общих свойств поля (одновременно на экране будут отображаться только те категории, которые соответствуют данному типу данных) приведен в табл. 6.4.

Таблица 6.4. *Краткий перечень общих свойств поля*

Название свойства	Описание
Вкладка **Общие**	
Размер поля	Если выбран текстовый тип данных, то это свойство позволяет ограничивать размер поля заданным числом символов. Для числовых типов данных можно задать тип числа

Таблица 6.4 *(продолжение)*

Название свойства	**Описание**
Вкладка **Общие**	
Формат поля	Указывает способ отображения на экране текста, чисел, значений дат и времени. Одни типы данных имеют стандартный формат, другие — формат, определенный пользователем, третьи допускают использование как стандартных, так и пользовательских форматов данных. Форматы влияют лишь на отображение данных, а не на способ их ввода или сохранения
Число десятичных знаков	Определяет количество десятичных знаков, выводимых после разделителя целой и дробной частей числа. Данное свойство доступно только для данных числового и денежного типа. Число десятичных знаков может изменяться от 0 до 15
Маска ввода	Задает шаблон, который облегчает ввод данных в поле. Используется при организации ввода данных, для которых необходим заданный формат (например, почтовые индексы, номера телефонов, даты и т. д.)
Подпись	Используется для отображения в формах или отчетах альтернативного имени для данного поля. Длина текста подписи — до 2028 символов. Нельзя использовать символ "*". Если подпись не задана, то используется заданное имя поля
Значение по умолчанию	Значение, которое появляется автоматически для данного поля при добавлении новой записи в таблицу. При задании значения по умолчанию можно использовать как некоторые заданные значения, так и значения, определяемые выражением (с помощью построителя выражений)
Условие на значение	Задаются требования к данным, вводящимся в поле. Условия на значение базируются на основе правил, созданных с помощью выражений (если используются сложные выражения, то рекомендуется их создавать с помощью построителя выражений) или макросов

Таблица 6.4 *(продолжение)*

Название свойства	**Описание**
Вкладка **Общие**	
Сообщение об ошибке	Текстовое сообщение, которое выводится на экран, если вводимые данные не соответствуют условиям ввода
Обязательное поле	Определяет, обязателен ли ввод значений в данном поле
Пустые строки	Определяет ввод значений " " в поле текстового типа для того, чтобы отличить его от пустого (null) значения
Индексированное поле	Ускоряет доступ к данным и ограничивает при необходимости вводимые данные только уникальными значениями
Сжатие Юникод	Используется для многоязычных приложений. По умолчанию это свойство принимает значение "Да", означающее, что все символы, первый байт которых в кодировке Юникод равен 0, будут сжиматься до одного байта при сохранении и восстанавливаться при выборке
Режим IME	Существует возможность задать параметры, управляющие работой Input Method Editor (IME — редактор метода ввода) для поля таблицы или элемента управления формы. IME — программа, обеспечивающая ввод текста на восточноазиатских языках (китайский с традиционным письмом, китайский с упрощенным письмом, японский и корейский). IME рассматривается как дополнительный вид раскладки клавиатуры
Режим предложений IME	После добавления смарт-тега к полю или элементу управления при активизации ячейки этого поля или элемента управления появляется кнопка  (действия для смарт-тегов), которая предлагает меню действий, доступных для смарт-тегов. Смарт-теги можно присоединять к файлу в таблице или запросе, к элементу управления в форме, отчете или на странице доступа к данным
Смарт-теги	

Таблица 6.4 *(окончание)*

Название свойства	**Описание**
Вкладка **Подстановка**	
Тип элемента управления	Определение типа элемента управления, который используется по умолчанию для отображения данного поля в формах, отчетах и объектах в режиме таблицы. Как правило, для большинства полей используется элемент управления — поле. Если поле является внешним ключом, то элементом управления будет список либо поле со списком. Если поле должно содержать заданный список значений, то также используются список и поле со списком
Тип источника строк	Задает тип источника данных для элемента управления
Источник строк	Конкретное указание на таблицу или запрос, из которых выбираются значения для списка. Если список данных вводится с клавиатуры, то значения разделяются точкой с запятой (;)
Присоединенный столбец	Используются значения столбца, из которого берутся данные. Если источник данных содержит один столбец, то вводится 1
Число столбцов	Количество столбцов, которые будут доступны из источника данных
Заглавия столбцов	Если задано "Да", то значения из источника данных выводятся с заголовками столбцов
Ширина столбцов	Задается значение для ширины столбцов
Число строк списка	Определяется количество строк, выводимых в раскрывающемся списке
Ширина списка	Определяется ширина раскрывающегося списка
Ограничиться списком	Если задано значение "Да", то в поле вводятся только данные из имеющихся значений списка

Использование маски ввода

Как указано в табл. 6.4, для облегчения ввода данных в поле можно использовать ввод по образцу, т. е. задание некоторого шаблона (маски) ввода. В маску ввода включаются различные специальные символы (табл. 6.5), которые используются только при выводе на экран.

Таблица 6.5. *Символы, используемые при задании маски ввода*

Символ	Описание
0	Вводятся цифры (обязательный символ); запрет на ввод символов "+" и "–"
9	Вводятся цифры или пробелы (необязательный символ); запрет на ввод символов "+" и "–"
#	Вводятся цифры, пробелы, символы "+" и "–" (необязательный символ; незаполненные позиции выводятся как пробелы в режиме редактирования, но удаляются при сохранении данных)
L	Буква (обязательный символ)
?	Буква (необязательный символ)
A	Буква или цифра (обязательный символ)
a	Буква или цифра (необязательный символ)
&	Любой символ или пробел (обязательный символ)
C	Любой символ или пробел (необязательный символ)
<	Все введенные после "<" символы преобразуются в строчные
>	Все введенные после ">" символы преобразуются в прописные
!	Указывает, что маска ввода заполняется слева направо. Рекомендуется использовать в случае, если в левой части маски находятся позиции, заполнять которые необязательно
\	Символ, следующий за символом "\", будет отображен в таком виде, в котором он был введен

Таблица 6.5 *(окончание)*

Символ	Описание
. , : ; – /	Десятичный разделитель, разделители групп разрядов, времени или даты
Пароль	Установка значения **Пароль** для свойства **Маска ввода** создаст текстовое поле ввода с паролем, в котором любой введенный в него символ будет сохраняться, но отображаться в виде символа "*"

Маска ввода может состоять из трех частей, разделяемых точкой с запятой ";".

- Непосредственное задание маски ввода; восклицательный знак в конце (например, `#0-000!`) указывает на то, что маска ввода заполняется слева направо.
- Указание, должны ли сохраняться введенные промежуточные символы в таблице, например, дефис или скобка (значение **0** — происходит сохранение промежуточных символов, значение **1** или пустое значение — символы не сохраняются).
- Указание символа, который будет использован вместо пробелов.

Пример.

Маска ввода, заданная шаблоном

```
\(##0)-#0\-00\-00;0;"_"
```

будет отображена в следующем виде в поле ввода

```
(___)-__-__-__
```

и приведет к указанному ниже отображению введенного числа:

```
(5)- 8-67-89
```

При формировании маски ввода удобно использовать мастер ввода маски. При щелчке на свойстве **Маска ввода** в бланке

Свойства поля (вкладка **Общие**) окна **Конструктор таблиц** появляется кнопка построителя масок [...] и вызывается мастер ввода маски (рис. 6.16).

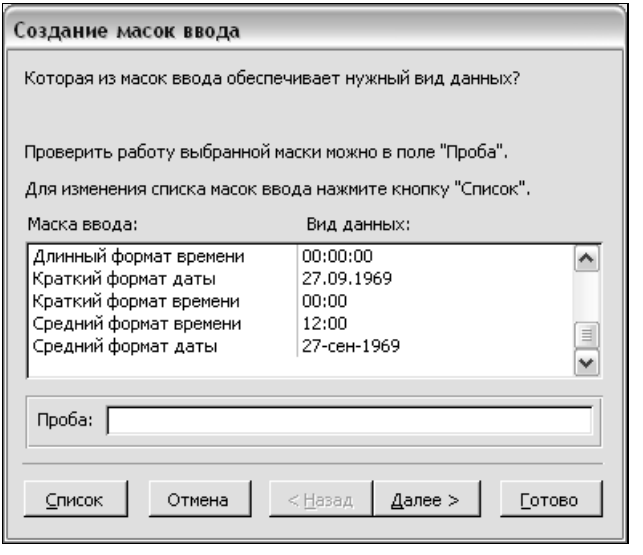


Рис. 6.16. Диалоговое окно **Создание масок ввода**

Выбор первичного ключа

Если таблица не будет использоваться в качестве главной, то ключ для нее определять не надо. В главных таблицах обычно содержится информация о реальных объектах, с которыми ассоциируется только одна запись.

В главной таблице связи должен быть определен первичный ключ. Ключ желательно определить и в связанных таблицах для предотвращения появления повторяющихся данных. При этом ключ таблицы также можно задать по значению нескольких полей.

Чтобы определить первичный ключ таблицы и проиндексировать ее по значению ключа, необходимо:

1. Открыть таблицу в режиме конструктора.
2. Выделить одно или несколько полей, которые выбраны ключевыми.
3. Нажать кнопку  **(Ключевое поле)** на панели инструментов **Конструктор таблиц** либо воспользоваться контекстным меню для выделенных полей, либо командой **Индексы** в меню **Вид**.
4. Чтобы определить последовательность, в которой выделенные поля входят в ключ, необходимо воспользоваться кнопкой **Индексы**  на панели инструментов **Конструктор таблиц**, и в открывшемся диалоговом окне указать последовательность полей (рис. 6.17).

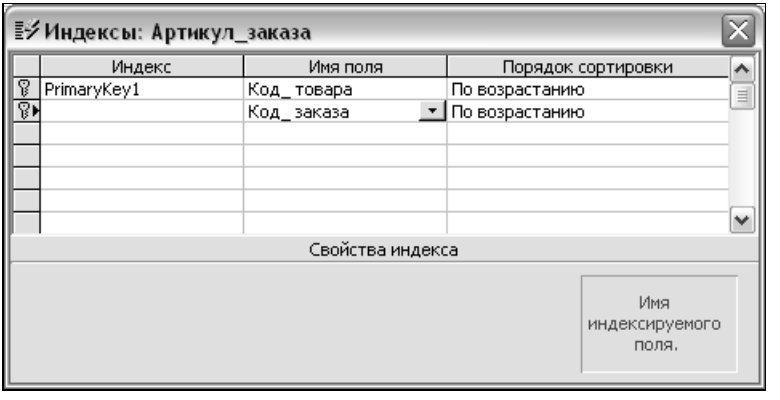


Рис. 6.17. Диалоговое окно **Индексы**

Индексирование таблицы

MS Access автоматически индексирует таблицу по значению ключа. Однако может потребоваться создание дополнительных индексов по значениям других полей. Индексы позволя-

ют ускорить поиск данных в тех полях таблицы, по которым она проиндексирована. Каждая таблица MS Access может иметь до 32-х индексов, 5 из них могут быть составными (в составной индекс может входить до 10-ти полей). Не следует создавать индексы для каждого поля таблицы и всех их комбинаций, т. к. это может существенно замедлить процесс заполнения таблицы (при изменении или добавлении записи автоматически обновляются все индексы).

Чтобы проиндексировать таблицу, необходимо:

1. Открыть таблицу в режиме конструктора.
2. Выделить необходимое поле в бланке структуры таблицы.
3. В нижней части окна конструктора установить значение "Да" для свойства **Индексированное поле** (на вкладке **Общие** в разделе **Свойства**) и выбрать из списка способ индексирования.

Рис. 6.18. Задание составного индекса

4. Для задания составного индекса открыть окно **Индексы** (нажать кнопку  на панели инструментов **Конструктор таблиц**), а затем в поле **Индекс** ввести имя индекса и последовательно задать имена полей, входящих в составной ин-

декс (рис. 6.18). При необходимости можно определить порядок сортировки для каждого поля составного индекса.

Создание схемы данных

Созданные таблицы, в которых определены поля, ключевые поля и индексы, связываются в *схему данных* с помощью графического окна **Схема данных** (команда **Схема данных** в меню **Сервис** либо кнопка  на панели инструментов **База данных**), показанного на рис. 6.19.

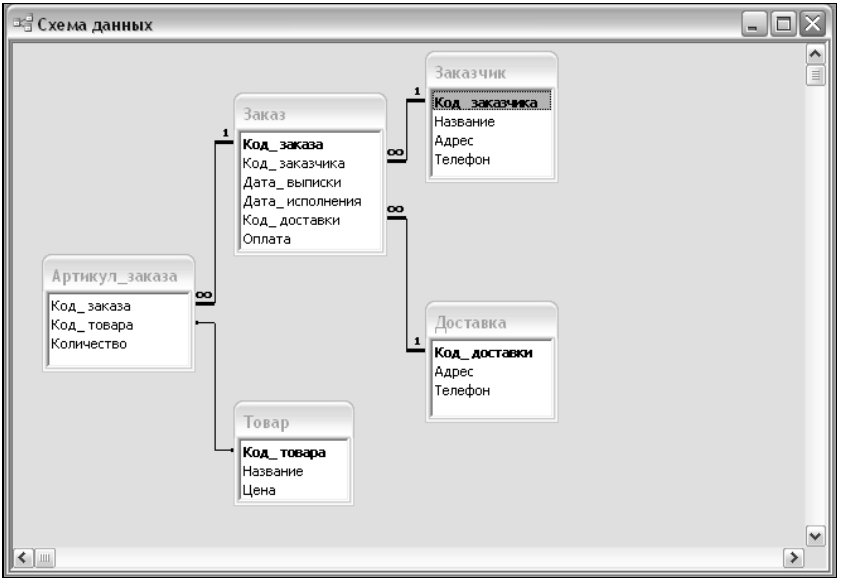


Рис. 6.19. Окно **Схема данных**

Чтобы добавить таблицу в схему, используют команду **Добавить таблицу** в меню **Связи** либо кнопку  на панели инструментов **Связь**. Связи между таблицами устанавливаются по значению полей. При создании новой связи очень важна последовательность действий: с помощью мыши поле со сторо-

ны "один" связи "один-ко-многим" перетаскивается на сторону "многие". В результате выполнения этого действия появляется диалоговое окно **Изменение связей** (рис. 6.20), в котором можно установить вид объединения двух таблиц и обеспечить целостность данных.

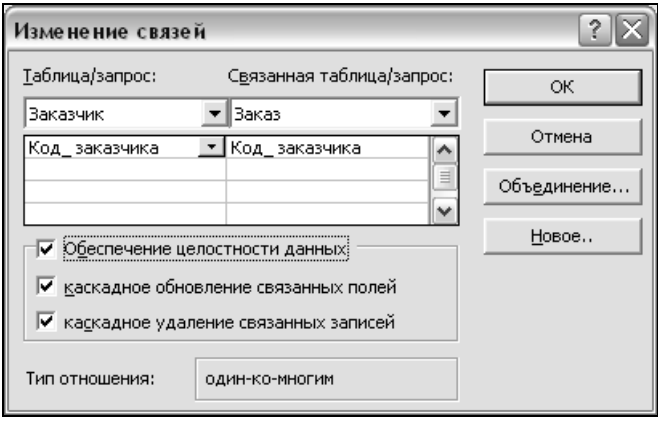


Рис. 6.20. Диалоговое окно **Изменение связей**

Установить необходимые параметры объединения таблиц можно в диалоговом окне **Параметры объединения**, нажав кнопку Объединение... в окне **Изменение связей** (рис. 6.21).

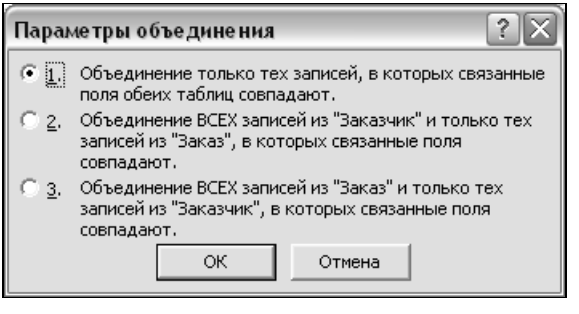


Рис. 6.21. Диалоговое окно **Параметры объединения**

Если на связь между таблицами наложены условия ссылочной целостности, то MS Access не позволяет добавлять в связанную таблицу записи, для которых нет соответствующих записей в главной таблице. Кроме того, нельзя изменять записи в главной таблице таким образом, чтобы в связанной таблице появились записи, не имеющие главных записей. Также нельзя удалять записи в главной таблице, для которых имеются подчиненные записи в связанной таблице. Условия целостности данных определяют систему правил, используемых MS Access для поддержания связей между записями в связанных таблицах. Эти правила делают невозможным случайное удаление или изменение связанных данных. После того как заданы условия целостности данных, на операции со связанными таблицами накладываются ограничения:

- невозможно ввести в поле внешнего ключа связанной таблицы значение, которое не существует в ключевом поле главной таблицы;
- нельзя также удалить запись из главной таблицы, если существуют связанные с ней записи в подчиненной таблице;
- запрещено изменение значения ключевого поля в главной таблице, если имеются записи, связанные с данной записью.

При наложении условий целостности данных любая попытка выполнить действие, нарушающее перечисленные запреты, приведет к выводу окна с предупреждением, а само действие выполнено не будет.

Режимы каскадного удаления записей и каскадного обновления данных таблиц, на которые наложены условия ссылочной целостности, позволяют упростить обновление и удаление данных из связанных таблиц при сохранении ссылочной целостности данных. При установленном параметре **Каскадное обновление связанных полей** (рис. 6.20) изменение значения в ключевом поле главной таблицы приводит к автоматическому обновлению соответствующих значений во всех связанных записях. При установленном параметре **Каскадное удаление связанных записей** удаление записи в главной таблице приводит к автоматическому удалению связанных записей в подчиненной таблице.

Изменение свойств полей и связей между таблицами

При разработке БД (например, на этапе тестирования) часто бывает необходимо изменить свойства полей и связей между таблицами. В этом случае следует придерживаться следующих рекомендаций.

- Чтобы изменить размер и тип данных полей, а также условие на значение, следует открыть таблицу в режиме конструктора и произвести необходимые изменения. Изменение типа данных может привести к их потере, поэтому следует учитывать, что:
  - изменение размера числового поля с меньшего на больший обычно происходит без потери точности данных; если же размер числового поля изменяется с большего на меньший, то MS Access производит усечение данных. Например, преобразование текстового типа к типу поля `MEMO` не влечет потери данных, а обратное преобразование усекает данные поля `MEMO` до 255 символов либо еще меньшего размера;
  - ни один из типов данных нельзя изменить на тип счетчик;
  - порядок следования полей в таблице можно изменить перетаскиванием в режиме таблицы либо в режиме конструктора.
- Для изменения связей между таблицами следует:
  - закрыть все окна таблиц и перейти в режим схемы данных;
  - щелкнуть правой кнопкой мыши по связи, которую нужно изменить, а затем выполнить необходимые изменения в окне **Изменение связей** (см. рис. 6.20).

Ввод и редактирование данных в таблицах

После того как в базе созданы все необходимые таблицы и связаны в схему данных, можно приступать к заполнению

данными БД. Следует заметить, что наиболее гибкий способ просмотра, редактирования, добавления и удаления данных обеспечивают различные формы. Однако в таблице, открытой в режиме таблица, также можно производить ввод и изменение данных.

Необходимо помнить, что при заполнении и модификации БД, будут учтены все ограничения, поэтому изначально рекомендуется осуществлять ввод данных в родительские таблицы (в которых определены наиболее существенные объекты предметной области и которые имеют уникальные идентификаторы — первичные ключи), а затем — в дочерние.

Для того чтобы осуществить ввод данных в таблицу, следует выделить таблицу в окне БД, перейдя к объектам 

, и воспользоваться, например, кнопкой 

в окне БД либо контекстным меню таблицы.

В пустой таблице отображается одна пустая запись, заполнение происходит последовательным вводом данных в ячейки таблицы. После того как введена хотя бы одна запись в таблицу, активная строка отмечается треугольником , расположенным в левом столбце от записи, а новая — звездочкой  .

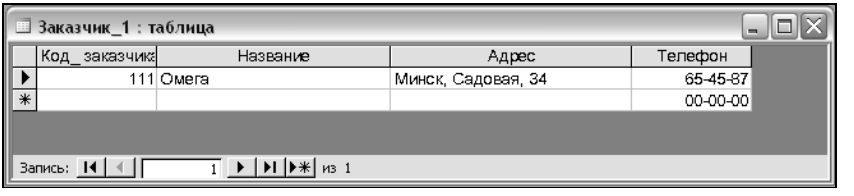


Рис. 6.22. Ввод данных в таблицу в режиме таблицы

Для ввода данных в таблицу необходимо установить курсор в нужной ячейке и ввести значения с клавиатуры. Однако следует помнить о том, что различный тип данных предполагает и их различное добавление и изменение в таблице. Так, например, поле с типом данных счетчик формируется MS Access

автоматически, а для поля с типом данных гиперссылка следует указать местоположение ресурса и т. д.

Перемещение по таблице осуществляется с помощью клавиш управления курсором либо щелчком левой кнопки мыши, можно также использовать кнопки перехода по записям, расположенные внизу окна таблицы (рис. 6.22).

После ввода данных окно таблицы закрывается, а содержимое таблицы сохраняется автоматически. Для сохранения вводимых значений в процессе заполнения содержимого таблицы можно воспользоваться командами категории линейки меню **Файл** либо кнопкой **Сохранить** на панели инструментов **Таблица в режиме таблицы**.

Замечание

Операцию сохранения можно произвести для любого выбранного объекта БД.

В MS Access можно удалять, копировать, перемещать как строки, так и отдельные значения ячеек.

Использование выражений

Выражение является основным средством выполнения многих операций MS Access и представляет собой комбинацию математических и логических операторов, констант, функций, имен полей, элементов управления и свойств, в результате обработки которой получается единственное значение. Выражение может выполнять вычисления, обрабатывать текст или проверять данные.

Выражения используются в следующих случаях:

- задание значения свойства, которое определяет для поля значение по умолчанию;
- задание условий отбора или обновления записей в запросе или фильтре;

- задание условий выполнения макросов;
- определение аргументов для многих функций и методов в процедурах VBA;
- при записи запроса на языке SQL.

Выражение обычно состоит из операторов сравнения и операндов (значений). Если выражение не содержит оператора, то по умолчанию используется оператор равенства "=". При формировании сложных условий возможно также использование логических и других операторов, указанных в табл. 6.6.

Таблица 6.6. *Основные операторы, используемые при создании выражений*

Оператор	Описание
>	Больше
>=	Больше или равно
=	Равно
<=	Меньше или равно
<	Меньше
<>	Не равно
&	Оператор слияния строк (позволяет объединить значение поля, элемента управления или свойства со строкой в явном представлении)
OR	ИЛИ
AND	И
NOT	Отрицание
IS	При использовании вместе с NULL определяет, является ли значение NULL (IS NULL) или NOT NULL (IS NOT NULL)
выражение [Not] In(значение1; значение2; …)	Проверяет равенство между выражением и значением из списка. Возможно использование с оператором NOT

Таблица 6.6 *(окончание)*

Оператор	Описание
`выражение [Not] Between значение1 And значение2`	Определяет, попадает ли значение выражения в указанный интервал, границы которого определяются величинами `значение1` и `значение2`. Возможно использование с оператором `NOT`
`выражение Like "шаблон"`	Сравнивает строковое выражение с шаблоном. В шаблоне можно либо указывать значение целиком (например, `Like "Сидоров"`), либо использовать подстановочные знаки, чтобы найти значения в некотором интервале (например, `Like "Сид*"`). К символам шаблона, используемым в операторе `Like`, относят: `?` — один неизвестный символ; `*` — любое количество неизвестных символов; `#` — одна произвольная цифра; `[ список ]` — один любой символ, который входит в список; `[! список ]` — один любой символ, который не входит в список

При создании выражений в качестве операндов могут использоваться различные значения: литералы, константы, функции, идентификаторы для указания значений в выражении, идентификаторы полей таблиц и др.

- *Константы* применяются для создания значений по умолчанию и для сравнения значений в таблицах.
- *Идентификаторы* — это имена объектов в MS Access (таких как поле таблицы), которые возвращают определенные числовые или текстовые значения.
- *Функции* возвращают значение, в выражениях аналогичны идентификаторам. В MS Access и VBA определено более 140 различных функций, которые можно использовать при создании собственных выражений.

Работа с операндами подразумевает использование некоторых специальных символов. Так, при работе с текстовыми строками, необходимо применять кавычки: вся используемая в выражении строка заключается в символы кавычек. Если в качестве операнда используется дата, то она ограничивается символами диеза "**#**" (например, `#12/02/05#`). Многие идентификаторы, которые представляют собой ссылку на значение поля, элемента управления или свойства, используют разделительный восклицательный знак "!" (указывает, что следующий за ним элемент является элементом, определяемым пользователем) или точку "." (обычно указывает, что следующий за ней элемент определен в MS Access). Имена таблиц, отчетов, запросов, полей и элементов управления заключаются в квадратные скобки "[ ]". При вводе имен объектов в идентификаторы использование квадратных скобок является обязательным, если имя содержит пробелы или специальные знаки, например, знак подчеркивания. Имена, не содержащие пробелы и специальные знаки, можно вводить без квадратных скобок. MS Access добавит скобки автоматически.

Выражения можно создавать самостоятельно или с помощью построителя выражений.

Самостоятельный ввод выражений возможен, например:

- непосредственно при задании свойств полей таблицы, в поле при формировании бланка запроса;
- конструировании того либо иного поля в форме или отчете;
- в ячейку аргумента макрокоманды

и т. д.

Однако использование построителя выражений существенно облегчает создание и редактирование выражений. Для того чтобы вызвать построитель выражения, достаточно воспользоваться кнопкой  , расположенной на любой панели инструментов **Конструктор** (**Конструктор таблиц**, **Конструктор запросов**, **Конструктор форм**, **Конструктор отчетов**, **Конструктор макросов**). Следует заметить, что возможности построителя выражений расширяются либо сужаются в зависимости от того, при редактировании и создании каких объектов он используется.

Чаще всего построитель выражений используется при формировании бланка запроса и состоит из трех разделов, располагающихся сверху вниз (рис. 6.23).

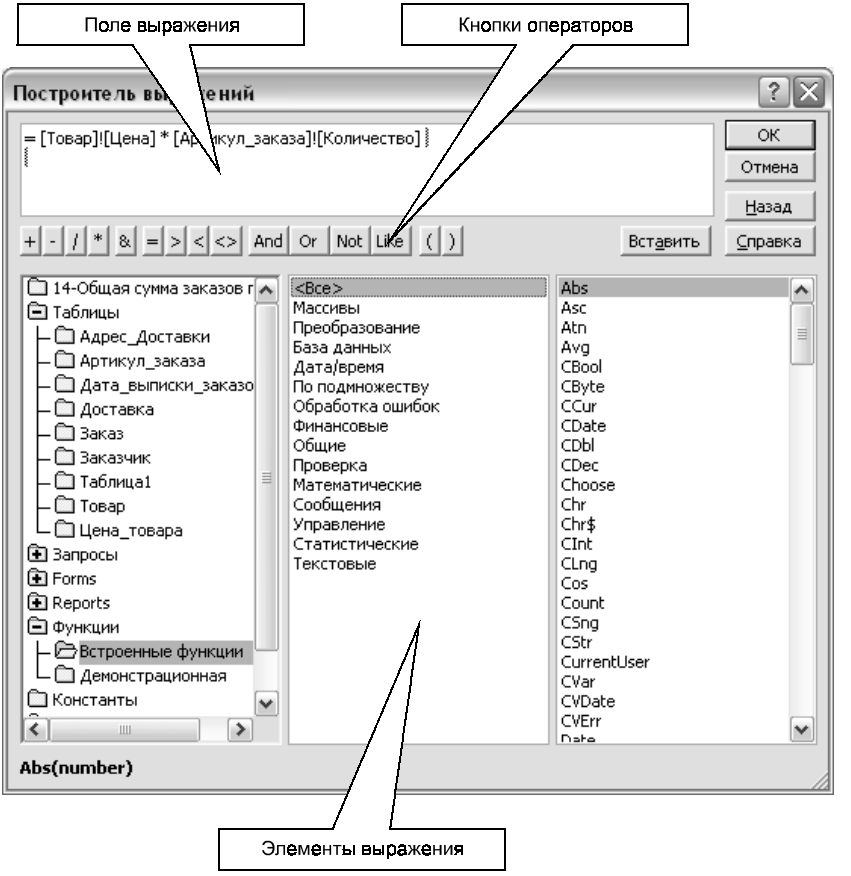


Рис. 6.23. Окно **Построитель выражений**

Поле выражения расположено в верхней части окна построителя выражений, в нем формируется требуемое выражение.

Кнопки операторов находятся в средней части окна. При нажатии на одну из этих кнопок построитель вставляет соответствующий оператор в текущую позицию поля выражения. Для вывода полного списка операторов следует выбрать папку **Операторы** в нижнем левом поле и нужный тип в среднем поле. В правом поле будут отображаться все операторы выбранного типа.

Элементы выражения расположены в нижней части окна **Построитель выражений**. В левом поле выводятся папки, содержащие таблицы, запросы, формы, объекты БД, встроенные и определенные пользователем функции, константы, операторы и общие выражения. В среднем поле задаются определенные элементы или типы элементов для папки, указанной в левом поле. Например, если выбрать в левом поле **Встроенные функции**, то в среднем поле появится список всех типов функций MS Access. В правом поле выводится список значений (если они существуют) для элементов, заданных в левом и среднем полях.

Пример *(доставка товаров).*

Рассмотрим в качестве упрощенного примера БД, на которой в дальнейшем будут демонстрироваться различные возможности MS Access. Так как детальное изложение этапа проектирования выходит за рамки рассматриваемого материала, будем полагать, что необходимые стадии проектирования и согласования выполнены, и разработка объектов БД будет происходить в соответствии со следующими требованиями:

- некоторая фирма занимается реализацией и доставкой товаров;
- информация, с которой приходится сталкиваться фирме, касается имеющихся товаров и заказчиков. Следует также учесть, что адрес доставки товаров может не совпадать с адресом заказчика;

- один и тот же заказчик может в течение непродолжительного времени приобрести товары, которые проходят по различным накладным.

Спроектировать и реализовать БД, которая ведет учет всех торговых операций и сопровождает их соответствующими накладными.

Решение.

1. Создать базу данных, используя табл. 6.7—6.11.

Таблица 6.7. *Товар*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ товара	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; сообщение об ошибке — "Введите правильный код товара!"; обязательное поле — да	Используется для идентификации товара
Название	Текстовый	Размер поля — 100 символов; обязательное поле — да; индексированное поле — нет	Название товара
Цена	Денежный	Формат вывода: `# ##0,00" р."`; условие на значение — не могут быть отрицательными; сообщение об ошибке — "Цена не может быть отрицательной!"; обязательное поле — да; индексированное поле — нет	Денежный эквивалент товара

Таблица 6.8. *Заказчик*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ заказчика	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; сообщение об ошибке — "Введите правильный код заказчика!"; обязательное поле — да	Используется для идентификации заказчика
Название	Текстовый	Размер поля — 100 символов; обязательное поле — да; индексированное поле — нет	Название организации либо фамилия заказчика
Адрес	Текстовый	Размер поля — 100 символов; обязательное поле — да; индексированное поле — нет	Адрес заказчика
Телефон	Числовой	Размер поля — длинное целое; формат вывода: `00-00-00;0`; маска ввода — `#00\-00\-00`; обязательное поле — нет; индексированное поле — нет	Телефон заказчика

Таблица 6.9. *Доставка*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ доставки	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; сообщение об ошибке — "Введите правильный код доставки!"; обязательное поле — да	Используется для идентификации доставки

Таблица 6.9 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Адрес	Текстовый	Размер поля – 100 символов; обязательное поле — да; индексированное поле — нет	Адрес доставки
Телефон	Числовой	Размер поля — длинное целое; формат вывода: `00-00-00;0`; маска ввода — `#00\-00\-00`; обязательное поле — нет; индексированное поле — нет	Телефон доставки

Таблица 6.10. *Заказ*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ заказа	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; сообщение об ошибке — "Введите правильный код заказа!"; обязательное поле — да	Используется для идентификации заказа
Код_ заказчика	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.8	Внешний ключ к табл. 6.8
Дата_ выписки	Дата/ время	Формат поля — краткий формат даты; обязательное поле — нет; индексированное поле — нет	Фиксируется дата оформления заказа

Таблица 6.10 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Дата_ исполнения	Дата/ время	Формат поля – краткий формат даты; обязательное поле — нет; индексированное поле — нет	Фиксируется дата исполнения заказа
Код_ доставки	Числовой, длинное целое	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.9	Внешний ключ к табл. 6.9
Оплата	Логический	Формат поля — да/нет; обязательное поле — нет; индексированное поле — нет	Фиксируется статус оплаты

Таблица 6.11. *Артикул заказа*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ заказа	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.10	Внешний ключ к табл. 6.10
Код_ товара	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.7	Внешний ключ к табл. 6.7

Таблица 6.11 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Коли-чество	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; сообщение об ошибке — "Количество не может быть отрицательным!"; обязательное поле — нет; индексированное поле — нет	Фиксируется количество проданных товаров

2. Связать таблицы в схему данных, используя связи "один-ко-многим", первичные и внешние ключи таблиц.
3. Создание таблиц (работа с бланком конструктора таблиц).

   Таблицы 6.7—6.11 создаются с использованием конструктора таблиц, для чего необходимо выполнить следующие действия:

   - для создания новой таблицы воспользоваться кнопкой 

в окне БД (в режиме объектов таблицы) либо выбрать команду **Создание таблицы в режиме конструктора** (см. рис. 6.12);
   - воспользоваться кнопкой  на панели инструментов **Конструктор таблиц** и ввести свойства, касающиеся всей таблицы в целом (рис. 6.24);
   - ввести в окно конструктора таблиц необходимые сведения, касающиеся имен полей, типов данных, свойств поля и описаний в соответствии с заданными значениями, приведенными в табл. 6.7—6.11 (рис. 6.25);
   - если в таблице необходимо определить первичные ключи, то следует выделить поле или поля, которые будут

ключевыми и воспользоваться контекстным меню либо кнопкой  панели инструментов **Конструктор таблиц**;

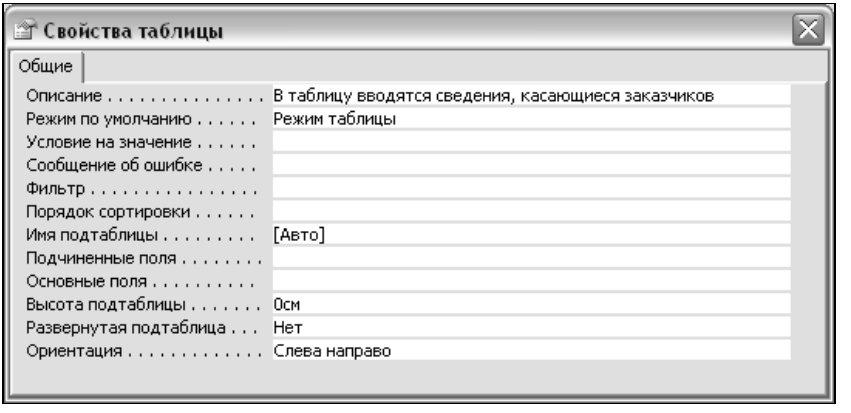


Рис. 6.24. Определение основных параметров таблицы в диалоговом окне **Свойства таблицы**

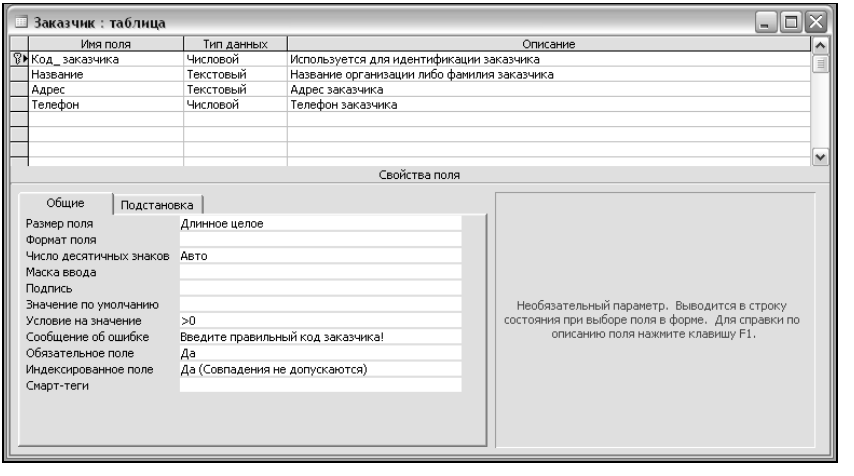


Рис. 6.25. Заполнение диалогового окна конструктора таблиц для создания новой таблицы

- задание индексов для таблицы происходит в диалоговом окне **Индексы**, которое открывается при нажатии кнопки **Индексы**  (рис. 6.26);

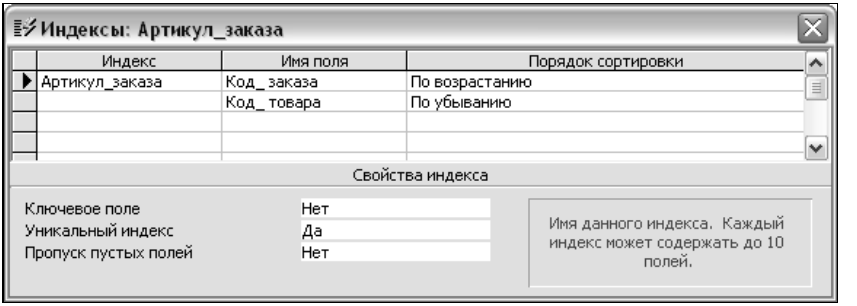


Рис. 6.26. Задание индекса для табл. 6.11

- если в процессе создания таблицы возникает необходимость редактирования либо удаления каких-либо значений, строк и т. д., можно воспользоваться соответствующими кнопками панели инструментов **Конструктор таблиц** (см. рис. 6.14) либо контекстным меню;
- после того как все необходимые сведения, касающиеся конкретной таблицы, введены, следует ее сохранить в БД, выполнив команду **Сохранить** в меню **Файл** и введя необходимое имя таблицы. При первом сохранении диалогового окна **Конструктор таблиц** появляется окно **Сохранение** (рис. 6.27).

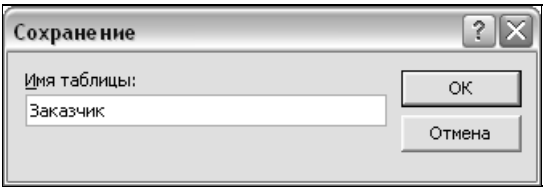


Рис. 6.27. Диалоговое окно **Сохранение**

4. Определение схемы данных.

   После того как определены все таблицы БД, следует определить схему БД в окне **Схема данных**. Для этого необходимо выполнить следующие действия:

   - воспользоваться командой **Схема данных** в меню **Сервис** либо кнопкой  на панели инструментов **База данных**;
   - используя диалоговое окно **Добавление таблицы** (команда **Добавить таблицу** в меню **Связи** либо кнопка  на панели инструментов **Связь**), добавить необходимые таблицы в окно **Схема данных** (рис. 6.28);

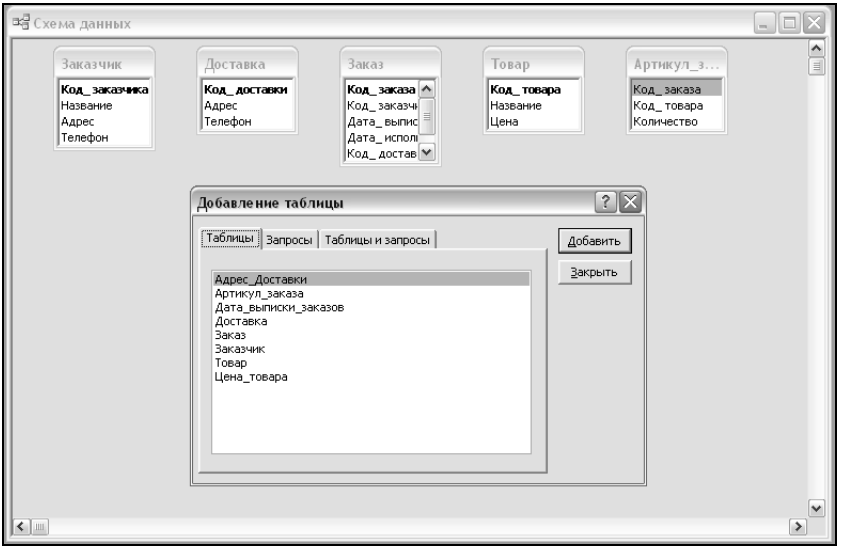


Рис. 6.28. Добавление таблицы в окно **Схема данных**

   - установить необходимые параметры связи между таблицами в диалоговом окне **Изменение связей** (рис. 6.29), перетягивая мышью ключевое поле родительской таблицы на соответствующее поле (внешний ключ) дочер-

ней таблицы. Можно также воспользоваться командой **Изменить связь** в меню **Связи**;

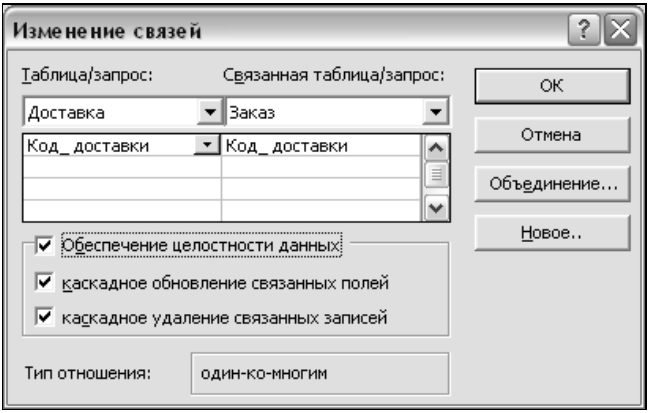


Рис. 6.29. Задание параметров связи в диалоговом окне **Изменение связей**

- после задания всех связей в окне **Схема данных** (см. рис. 6.19) можно определить поля подстановки для дочерних таблиц. Для этого, не закрывая окна **Схема данных**, следует перейти в режим конструктора дочерней таблицы (в данном случае это табл. 6.10 и 6.11) и выбрать внешние поля (в табл. 6.10 это поля `Код_заказчика` и `Код_доставки`; в табл. 6.11 — `Код_заказа` и `Код_товара`), для которых необходимо установить свойства на вкладке **Подстановка** (рис. 6.30);
- после внесения всех необходимых корректив в схему данных окно **Схема данных** закрывают и подтверждают сохранение сделанных изменений.

5. Заполнение базы данными.

   При вводе данных в созданные таблицы нужно придерживаться следующих рекомендаций:

   - вводить данные в режиме таблицы и последовательно заполнять каждую ячейку.

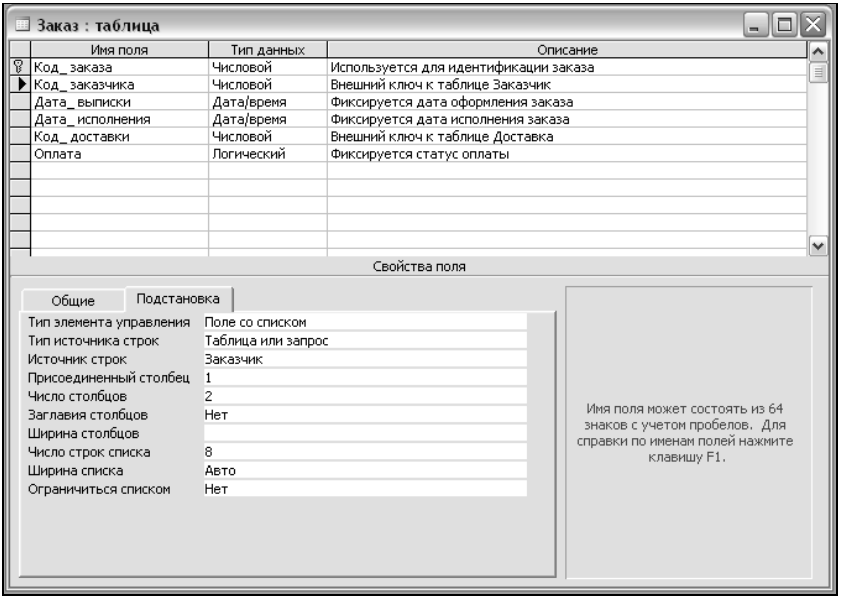


Рис. 6.30. Установка свойств поля на вкладке **Подстановка** для поля `Код_заказчика` табл. 6.10

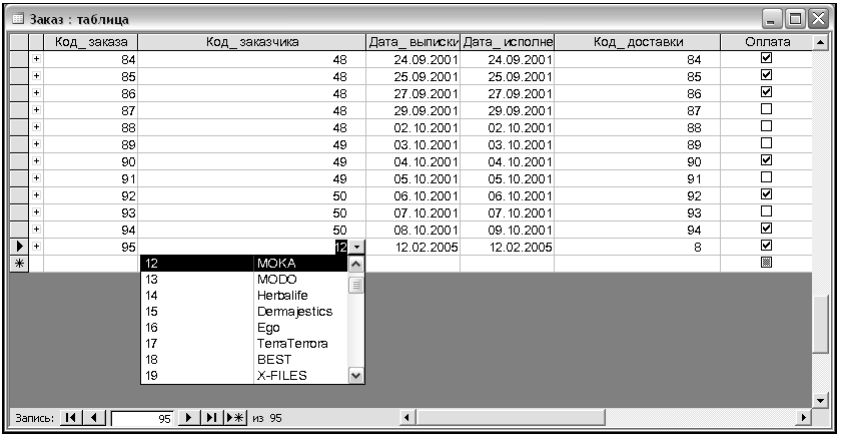


Рис. 6.31. Добавление записей в табл. 6.10

Если возникнет необходимость редактирования либо удаления данных, использовать соответствующие команды линейки меню, кнопки панелей инструментов либо контекстное меню для выделенного диапазона;

- внести данные сначала в родительские таблицы (табл. 6.7—6.9), а затем в дочерние: первоначально в табл. 6.10 (в ней фиксируется сделанный заказ), потом в табл. 6.11 (в ней хранится содержимое заказа). С учетом всех ограничений и созданных списков ввода заполнение БД не представляет трудностей (рис. 6.31).

Обработка данных средствами Microsoft Access

Обработка данных средствами MS Access включает в себя следующие направления:

- *сортировка* — выстраивание данных в нужном порядке;
- *поиск данных* — извлечение записей данных из списка в соответствии с некоторыми требованиями (критериями).

Сортировка, поиск и фильтрация данных

Средства сортировки, фильтрации, поиска и замены данных реализованы в MS Access как автоматически создаваемые запросы. Данные операции производятся с таблицами или запросами, открытыми в режиме таблицы. Сортировку и поиск данных можно выполнить, используя соответствующие команды:

- для сортировки записей следует применять команду **Сортировка** в меню **Записи** (далее необходимо выбрать вид сортировки — **По возрастанию** либо **По убыванию**);

- для отмены сортировки — воспользоваться командой **Удалить фильтр** в меню **Записи**.

При фильтрации отбор данных происходит из таблицы или запроса с учетом некоторого критерия отбора.

Различают фильтры трех видов.

- *Фильтр по выделенному* фрагменту. Критерием отбора в данном фильтре является значение или части значения поля таблицы. Это наиболее быстрый способ отбора данных. Недостаток данного вида фильтрации — отбор записей по значению только одного поля. Применение данного фильтра может происходить следующим образом:
  - выделить значение или часть значения поля в таблице (либо запросе);
  - выбрать в меню **Записи** пункт **Фильтр**, команду **Фильтр по выделенному** либо воспользоваться кнопкой  на панели инструментов **Таблица в режиме таблицы**;
  - при необходимости отмены результатов фильтрации выбрать в меню **Записи** пункт **Фильтр**, команду **Удалить фильтр**.
- *Обычный фильтр* — это отбор записей по значению нескольких полей. Для задания критерия отбора заполняется специальная форма (бланк). Для использования обычного фильтра следует:
  - открыть таблицу (либо запрос) в режиме таблицы;
  - выбрать в меню **Записи** пункт **Фильтр**, команду **Изменить фильтр** (кнопка  на панели инструментов **Таблица в режиме таблицы**); для того чтобы очистить предыдущие критерии отбора, следует выбрать команду **Очистить бланк** в меню **Правка** (либо кнопку  на панели инструментов **Фильтр**);
  - установить критерии отбора, используя логические операции И, ИЛИ, НЕ, а также различные выражения;

следует помнить, что если в одной из вкладок окна бланка критериев отбора заполнены критерии для нескольких полей, то критерии объединяются с помощью логической операции И;

- воспользоваться командой **Применить фильтр** в меню **Фильтр** (либо кнопкой  на панели инструментов **Фильтр**).

❑ *Расширенный фильтр* представляет собой отбор записей в соответствии с критерием отбора для различных полей таблицы, включающий сортировки по данным полям. Расширенный фильтр аналогичен расширенному фильтру в MS Excel. Окно расширенного фильтра MS Access отличается от окна запросов следующим:

- невозможно вызвать диалоговое окно **Добавление таблицы**;
- отсутствует переход в режим SQL;
- используется только одна таблица для отбора данных.

Для использования расширенного фильтра в MS Access необходимо:

- открыть таблицу (или запрос) в режиме таблицы;
- выбрать в меню **Записи** пункт **Фильтр**, команду **Расширенный фильтр**;
- установить критерии отбора, использующие необходимые условия и выражения;
- выбрать в меню **Записи** пункт **Фильтр**, команду **Применить фильтр** либо воспользоваться кнопкой  на панели инструментов **Фильтр**;
- результаты фильтрации можно сохранить в качестве запроса, используя команду **Сохранить как запрос** в меню **Файл**.

Запросы в Microsoft Access

Запросы представляют собой некоторый набор данных, полученный в результате обращения к БД. Запрос может включать различные условия, вычисляемые поля, инструкции на выполнение тех или иных операций и т. д.

Общие сведения о запросах в Microsoft Access

Запрос — это динамический (виртуальный) набор данных, которые существуют только во время выполнения запроса. В силу этого при каждом новом выполнении запроса можно получать различные результаты, учитывающие все изменения, которые произошли с данными в таблицах MS Access.

В MS Access предусмотрено создание запросов в двух режимах: в режиме QBE, использующем графический бланк для конструирования запроса (рис. 6.32), и SQL, позволяющий создавать инструкции на выборку данных с помощью встроенного языка SQL-запросов.

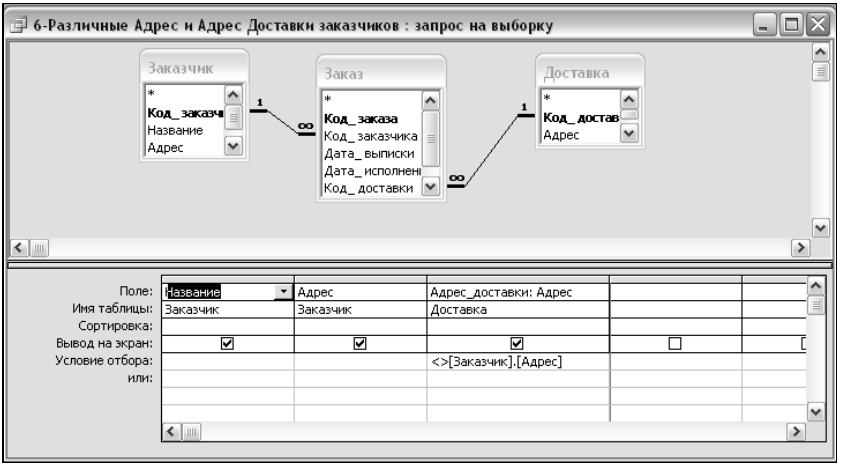


Рис. 6.32. Стандартное окно конструктора запросов

Основные виды запросов, которые достаточно часто используются в MS Access, приведены в табл. 6.12.

Таблице 6.12. *Основные виды запросов MS Access*

Тип запроса	**Описание**
Запрос на выборку	Выводит данные, находящиеся в одной или нескольких таблицах, в соответствии с некоторыми критериями. Результаты запроса представляются в виде таблицы, в которой допускается изменение записей. При создании запросов такого типа используется стандартный бланк (рис. 6.32)
Групповой запрос	Предусматривает выполнение вычислений с использованием данных некоторой группы записей. При выборе этого типа запроса в стандартный бланк запроса необходимо добавить поле **Групповая операция**
Перекрестный запрос	Выводит результаты статистических расчетов (например, среднее значение, сумма, количество записей) для данных, которые находятся в одной или нескольких двумерных таблицах. Как правило, результаты таких запросов используются для анализа данных и создания диаграмм. Для выполнения такого запроса в бланк запроса добавляются два поля: **Групповая операция** и **Перекрестная таблица**
Запрос на изменение (модифицирующий)	Запросы такого типа влияют на содержимое БД. Они позволяют за одну операцию внести изменения во множество записей. Используются для создания новых таблиц из результатов запроса и для внесения изменений в данные существующих таблиц. Различают запросы на удаление, добавление и обновление данных, а также запрос на создание таблиц. Для каждого вида запроса в бланк конструктора добавляется соответствующее поле (поля)
Параметрический запрос	Запросы, свойства которых изменяются пользователем при каждом запуске. Выполнение этих запросов сопровождается выводом одного или более диалоговых окон, предназначенных для ввода пользователем конкретных значений параметров запроса. Этот тип запроса явно не выражен, т. к. параметр можно добавить к запросу любого типа

Таблице 6.12 *(окончание)*

Тип запроса	Описание
SQL-запрос (включает функции соединения, передачи определенных данных, а также подзапросы)	В запросах такого типа применяются специфические средства языка SQL, например, операции соединения, операторы определения данных и подзапросы (подзапрос представляет собой запрос, встроенный в тело другого запроса), а также передаваемые запросы в СУБД SQL Server компаний Microsoft или Sybase. SQL-запросы в MS Access бывают трех видов: запрос на объединение, запрос к серверу и управляющий запрос

Следует также отметить, что все запросы, перечисленные в табл. 6.12, могут включать критерии отбора, вычисляемые выражения и группировки.

Если запрос включает несколько таблиц, то необходимо четко представлять все связанные поля этих таблиц, а также порядок действий, приводящий к конечному результату запроса. MS Access поддерживает четыре типа соединений.

- *Внутреннее соединение* (эквисоединение) — используется при создании запросов на выборку. Результат запроса содержит записи одной таблицы, имеющие совпадающие значения в связанных полях другой таблицы. Если в дочерней таблице отсутствуют записи (со стороны "многие"), то в результате запроса записи родительской таблицы (со стороны "один" не включаются).
- *Внешнее соединение* используется для создания запроса, в котором можно вывести данные одной из таблиц независимо от того, имеются ли соответствующие записи в другой таблице.
- *Рекурсивное соединение* связывает данные в одной таблице. Такое соединение получается путем добавления в запрос копии таблицы и связывания полей идентичных таблиц. Рекурсивные соединения используются очень редко в пол-

ностью автоматических БД, т. к. заданные условия на значения и обеспечиваемая целостность данных сводят на нет необходимость использования таких связей.

- *Соединение по отношению* (Θ-соединение) связывает данные некоторым отношением (за исключением равенства).

Рекомендации по созданию запросов в Microsoft Access

Запросы в MS Access можно создавать после того, как созданы таблицы, схема и добавлены данные в базу.

Для создания запросов необходимо перейти к объектам **Запросы** в левой части окна БД (рис. 6.33) и воспользоваться кнопкой Создать в окне БД.

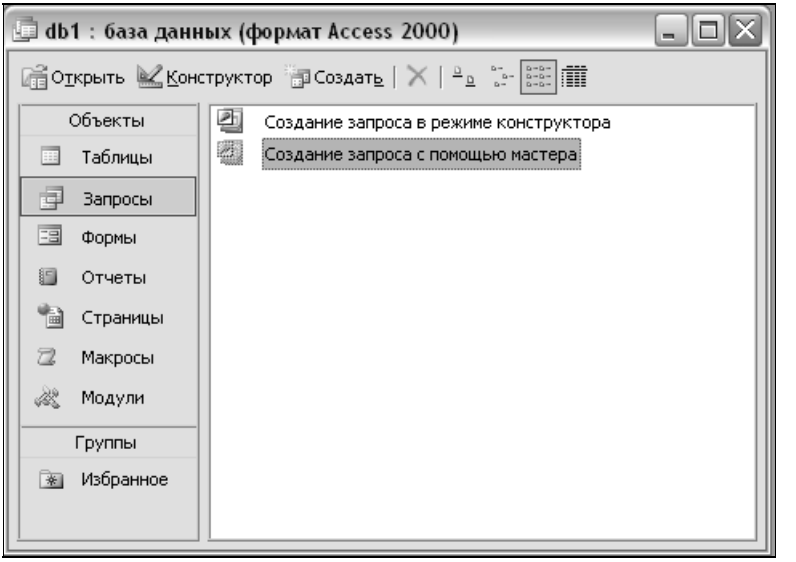


Рис. 6.33. Окно БД
в режиме отображения объекта **Запросы**

В открывшемся диалоговом окне **Новый запрос** можно выбрать одну из следующих возможностей создания запроса:

- **Конструктор** — самостоятельное создание нового запроса в графическом бланке QBE;
- **Простой запрос** — создание запроса с помощью мастера на выборку из определенных полей таблицы (таблиц);
- **Перекрестный запрос** — создание запроса с помощью мастера, результаты которого представлены в виде двумерной таблицы;
- **Повторяющиеся записи** — создание запроса с помощью мастера на поиск повторяющихся записей в простой таблице или запросе;
- **Записи без подчинений** — создание запроса с помощью мастера на поиск записей, которым не соответствует ни одна запись в подчиненной таблице.

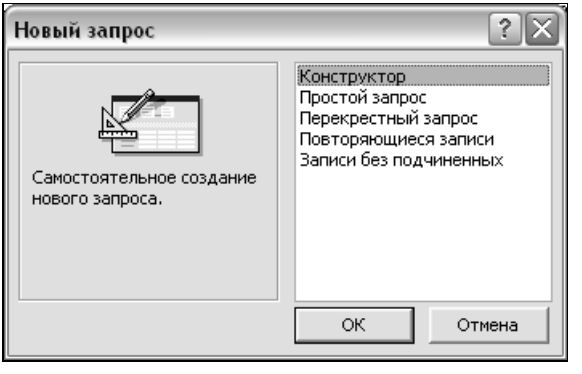


Рис. 6.34. Диалоговое окно **Новый запрос**

Работа по созданию запросов с помощью мастера сводится к последовательному выполнению инструкций и не представляет особых сложностей.

Диалоговое окно конструктора запроса (рис. 6.35) состоит из двух частей:

- *Область таблиц запроса* — место в верхней части окна конструктора запроса, в котором размещаются таблицы и/или запросы и отображается их структура;
- *Бланк запроса* — место в верхней части окна конструктора запроса, в котором задаются необходимые поля и условия запроса. В каждом столбце бланка содержится информация об одном поле таблицы или запроса, расположенном в области таблиц запроса.

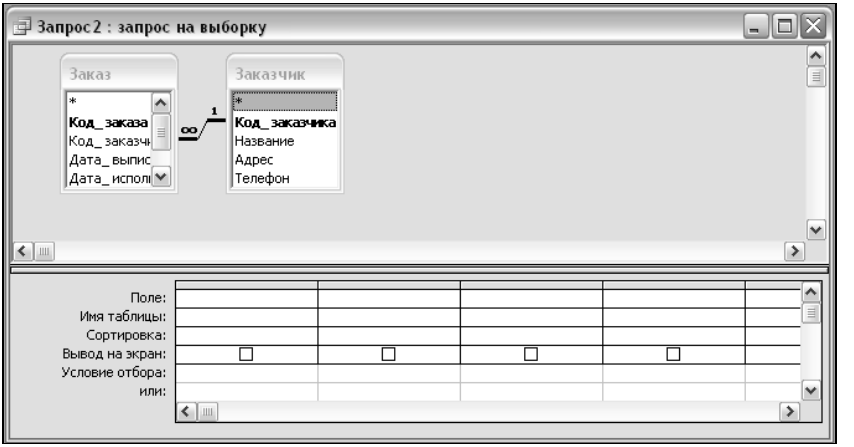


Рис. 6.35. Окно конструктора запроса с таблицами "Заказ" и "Заказчик"

С помощью соответствующей команды категории меню **Вид** либо кнопки на панели инструментов **Конструктор запросов** (рис. 6.36) окно конструктора запроса может переходить в различные режимы (рис. 6.37):

- режим конструктора — формирование запроса с помощью графического языка QBE;

- режим таблицы — выводится результат запроса;
- режим SQL — запрос отображается либо формулируется в виде инструкций на языке SQL;
- режим сводной диаграммы (таблицы) — предоставляется возможность создания сводной диаграммы (таблицы) на основе полей таблиц БД;

Рис. 6.36. Панель инструментов **Конструктор запросов**

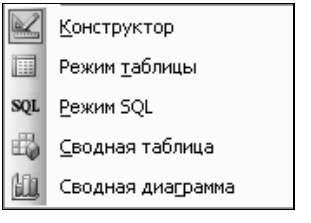


Рис. 6.37. Режимы отображения окна конструктора запросов

В область таблиц запроса всегда можно добавить необходимые объекты с использованием команды **Отобразить таблицу** в меню **Запрос** либо кнопки  на панели инструментов **Конструктор запросов** (при создании нового запроса в режиме конструктора диалоговое окно **Добавление таблицы** выводится по умолчанию).

Бланк запроса может содержать основные поля, представленные в табл. 6.13.

Таблица 6.13. *Основные поля стандартного бланка запроса*

Название	Описание
Поле	Имя поля
Имя таблицы	Имя таблицы, в которой содержится выбранное поле
Сортировка	Указывается вид сортировки (по возрастанию, по убыванию)
Вывод на экран	Определяет вывод на экран поля, указанного в бланке запроса
Условие отбора	Содержит первое условие, ограничивающее набор записей
Или	Другие условия ограничения для вывода записей

Как указывалось ранее, стандартный бланк запроса выводится на экран автоматически при выборе варианта создания запроса с помощью конструктора. В зависимости от того, запросы какого типа необходимо реализовать (см. табл. 6.12), в бланк запроса могут быть добавлены либо исключены те или иные поля.

Переход к необходимому виду запроса осуществляется с помощью соответствующих команд категории линейки меню **Запрос** либо с помощью кнопки  на панели инструментов **Конструктор запросов** (рис. 6.38).

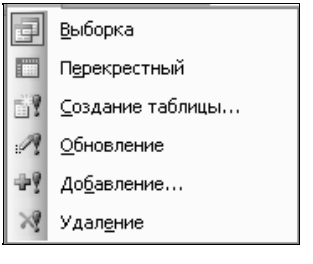


Рис. 6.38. Возможности создания различных типов запросов

Кроме того, команда **Групповые операции** в меню **Вид** (соответствующая кнопка ) позволяет добавить в бланк запроса поле "групповая операция".

Кнопка  расширяет возможности создания запросов: в окне построителя выражений можно конструировать вычисляемые поля и задавать сложные условия отбора записей (см., например, табл. 6.14).

Определить общие свойства для запроса в целом можно с использованием кнопки  , например, задать вывод уникальных значений либо уникальных записей. Следует помнить, что кнопка **Свойства** отображает возможности для всего запроса только в том случае, если произведен щелчок левой кнопкой мыши на строке заголовка запроса или в области таблиц запроса (вне таблиц).

После того как заданы необходимые критерии запроса, результат запроса можно вывести с помощью команды **Запуск** из меню **Запрос** (либо кнопки ).

Если инструкцию запроса необходимо сохранить в БД, то используют команду **Сохранить** в меню **Файл** (кнопка ) либо подтверждают операцию сохранения вводом имени запроса при закрытии соответствующего окна.

Итак, заполнение табличной формы конструктора запросов MS Access включает следующие этапы:

1. Выбор режима конструктора при создании запроса.
2. Определение свойств, относящихся к запросу в целом (например, с использованием кнопки ).
3. Определение типа запроса и включение таблиц, из которых производится выборка, в бланк запроса.
4. Определение и добавление полей таблиц, которые участвуют в запросе, в бланк запроса. При необходимости применяется переименование существующих полей и создаются новые вычисляемые поля.

Включение полей в бланк может осуществляться либо перетаскиванием мышью необходимого поля из таблицы, либо соответствующей выборкой нужного поля в бланке запроса.

Заголовок поля можно изменить в бланке запроса следующим образом: перейти к заголовку поля, которому необходимо присвоить новое имя, и перед старым названием вписать новое имя поля, разделив их знаком двоеточия (без пробелов):

```
Новое_название:Старое_название
```

В случае включения в бланк запроса вычисляемого поля следует после задания имени поля сразу же написать вычисляемое выражение (удобнее использовать при этом построитель выражений):

```
Имя_поля:Вычисляемое_выражение
```

Таблица 6.14. *Примеры выражений для создания вычисляемых полей*

Имя столбца	Выражение	Вычисленное значение
Стоимость товара	`[Товар]![Цена]* [Артикул_заказа]! [Количество]`	Произведение полей `Цена` из табл. 6.7 "Товар" и `Количество` из табл.6.11 "Артикул_заказа"
Налоговое отчисление	`FormatCurrency(Iif ([Товар]![Цена]* [Артикул_заказа]! [Количество]<600000; (0,1*[Товар]![Цена]* [Артикул_заказа]! [Количество]);(0,5* [Товар]![Цена]* [Артикул_заказа]! [Количество])))`	Вычисляет сумму налоговых отчислений со стоимости каждого проданного товара и представляет итоговое значение в денежном формате. На сумму отчислений налагаются условия: при стоимости товара менее 600 000 р. удерживается 10% суммы, при большей стоимости товара — более 50%

Таблица 6.14 *(окончание)*

Имя столбца	Выражение	Вычисленное значение
Общая сумма оплаты	`([Товар]![Цена]*[Артикул_заказа]![Количество])+FormatCurrency(Iif([Товар]![Цена]*[Артикул_заказа]![Количество]<600000;(0,1*[Товар]![Цена]*[Артикул_заказа]![Количество]);(0,5*[Товар]![Цена]*[Артикул_заказа]![Количество])))`	Вычисляет общую стоимость товара, включающую его чистую стоимость и налоговое отчисление

5. Задать (при необходимости) порядок сортировки (в поле **Сортировка бланка запроса**).
6. Указать (при необходимости) групповую операцию для какого-либо поля в случае проведения вычислений на множестве записей (поле **Групповая операция** в бланке).

   Типичными примерами использования групповых операций могут служить запросы общей стоимости заказа, стоимости всех проданных товаров с учетом их наименования и др. Основные функции, используемые в группировках, приведены в табл. 6.15.

Таблица 6.15. *Функции, используемые при групповых операциях в запросах*

Функция	Описание	Типы полей
`AVG()`	Среднее арифметическое набора чисел, содержащихся в указанном поле запроса	Все типы полей, исключая текстовые, поле `MEMO` и поле объекта OLE
`COUNT()`	Количество непустых записей запроса	Все типы полей

Таблица 6.15 *(окончание)*

Функция	Описание	Типы полей
FIRST()	Возвращает значение поля из первой записи результирующего набора	Все типы полей
LAST()	Возвращает значение поля из последней записи результирующего набора	Все типы полей
MAX()	Находит максимальное из набора значений, содержащихся в указанном поле	Все типы полей, исключая текстовые, поле MEMO и поле объекта OLE
MIN()	Находит минимальное из набора значений, содержащихся в указанном поле	Все типы полей, исключая текстовые, поле MEMO и поле объекта OLE
STDDEV() STDDEVP()	Возвращает смещенное и несмещенное значение среднеквадратичного отклонения, вычисляемого по набору значений, содержащихся в указанном поле	Все типы полей, исключая текстовые, поле MEMO и поле объекта OLE
SUM()	Сумма набора значений, содержащихся в указанном поле	Все типы полей, исключая текстовые, поле MEMO и поле объекта OLE
VAR() VARP()	Возвращает значение смещенной и несмещенной дисперсии, вычисленной по набору значений, содержащихся в указанном поле	Все типы полей, исключая текстовые, поле MEMO и поле объекта OLE

7. Определить условия (критерии) отбора записей. Критерии отбора записей вводятся в строке **Условие отбора** бланка конструктора запросов под конкретным полем запроса. Выбор записей в общем случае может быть основан на точном или частичном совпадении, сравнении и использовании блока условий. При задании критерия отбора следует использовать различные операторы, функции и выражения (табл. 6.16).

Таблица 6.16. *Примеры типичных выражений для критериев запроса*

Таблица	Поле	Выражение	Возвращаемое значение
6.8	Название	`NOT "Omega" AND NOT "Delta"`	Название фирм заказчиков кроме фирм Omega и Delta
6.8	Название	`"Omega" OR "Delta"`	Название фирм заказчиков Omega или Delta
6.8	Название	`LIKE "[А-Г]*"`	Фирмы заказчиков с названиями на буквы А—Г
6.8	Название	`LIKE "Б*" OR "З*"`	Фирмы заказчиков с названиями на буквы Б или З
6.8	Название	`LIKE "*ск*"`	Фирмы заказчиков с названиями, содержащими "ск"
6.11	Количество	`>= 100`	Количество наименований товара больше или равно 100
6.10	Дата_выписки	`YEAR([ДатаВыписки])=2005`	Накладные, выписанные за 2005 г.
6.10	Дата_выписки	`BETWEEN #01.09.04# AND #31.12.04#`	Накладные, выписанные в последнем квартале 2004 г.
6.10	Оплата	`IS NULL`	Неоплаченные накладные
6.7	Цена	`BETWEEN 10000 AND 25000`	Товары, цена которых находится в пределах от 10 000 до 25 000 р.

8. Определить параметры выборки запроса (нужно ли при выполнении запроса вводить некоторую уточняющую информацию). Как отмечалось ранее, параметрический запрос может быть совместим с запросом любого вида. Для

создания параметра в поле **Условие отбора бланка запроса** необходимо ввести текст сообщения в прямоугольных скобках (рис. 6.39).

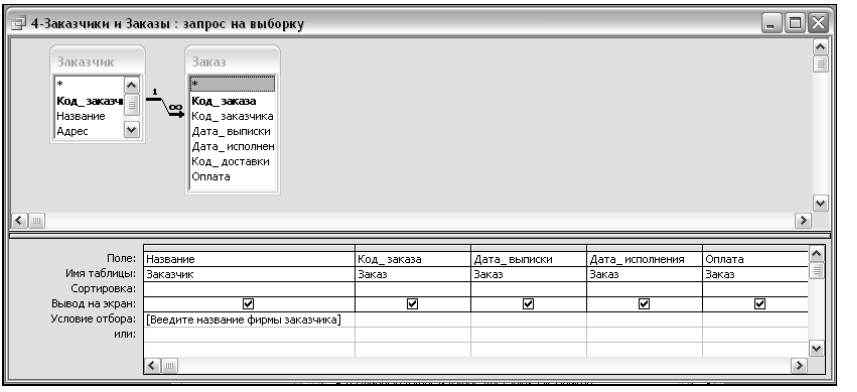


Рис. 6.39. Задание параметра ввода в строке **Условие отбора**

Каждый раз при запуске запроса, содержащего параметр, на экран будет выводиться соответствующее окно диалога (рис. 6.40).

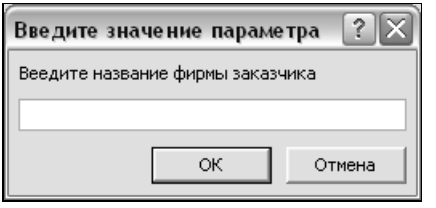


Рис. 6.40. Окно ввода параметра

Если параметров для выбранного запроса задано несколько, то их порядок можно изменить в диалоговом окне **Параметры запроса**.

9. В зависимости от вида выбранного запроса возможно также заполнение соответствующих полей для конкретного вида запросов.
10. Проверка запроса на выполнение и сохранение инструкции запроса.

Примеры запросов

Запросы на выборку

Запросы на выборку в окне БД при переходе к режиму запросов отображаются значком  .

Пример. Получить даты выписки накладных и даты исполнения для всех заказчиков (внутреннее соединение по одному полю).

Решение.

Результат решения примера приведен на рис. 6.41.

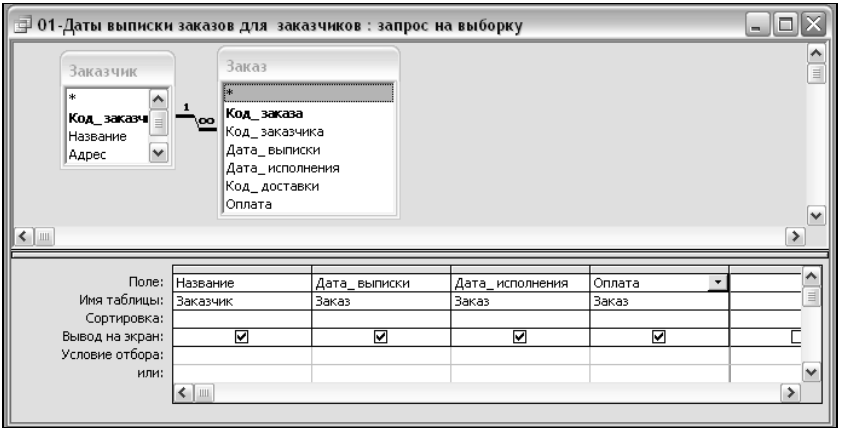


Рис. 6.41. Внутреннее соединение по одному полю

Для решения примера необходимо:

1. Перейти к созданию запроса в режиме конструктора (см. рис. 6.44).
2. С помощью диалогового окна **Добавление таблицы** внести в бланк запроса табл. 6.8 и 6.10.
3. Добавить в бланк запроса в поле **Имя** следующие поля: `Название` (из табл. 6.8), `Дата_выписки`, `Дата_исполнения`, `Оплата` (из табл. 6.10).
4. Для поля `Название` установить сортировку по возрастанию (в поле **Сортировка**).
5. В поле **Вывод на экран** проверить соответствующую отметку для всех полей, включенных в запрос.

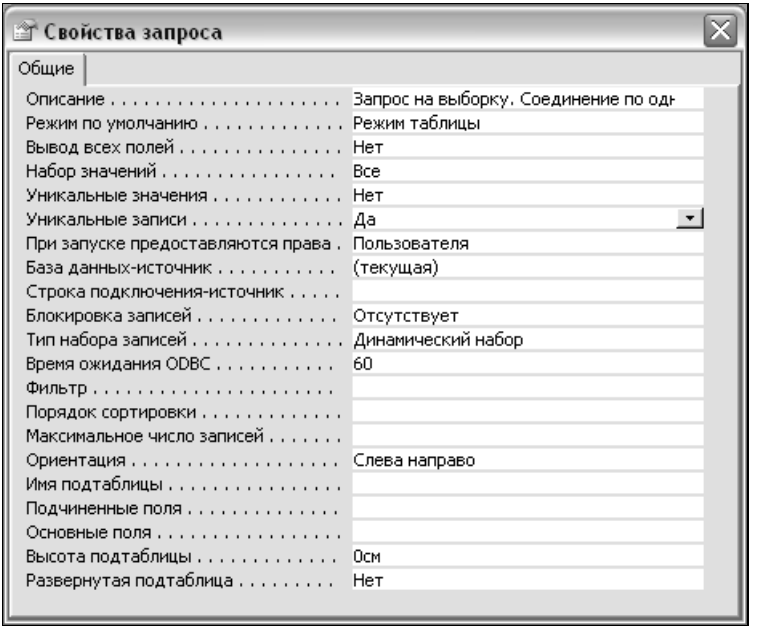


Рис. 6.42. Диалоговое окно **Свойства запроса**

6. Установить отметку **Да** в строке **Уникальные записи** в диалоговом окне **Свойства запроса** (рис. 6.42).

7. Выполнить проверку запроса (кнопка  на панели инструментов **Конструктор запроса**, рис. 6.43).

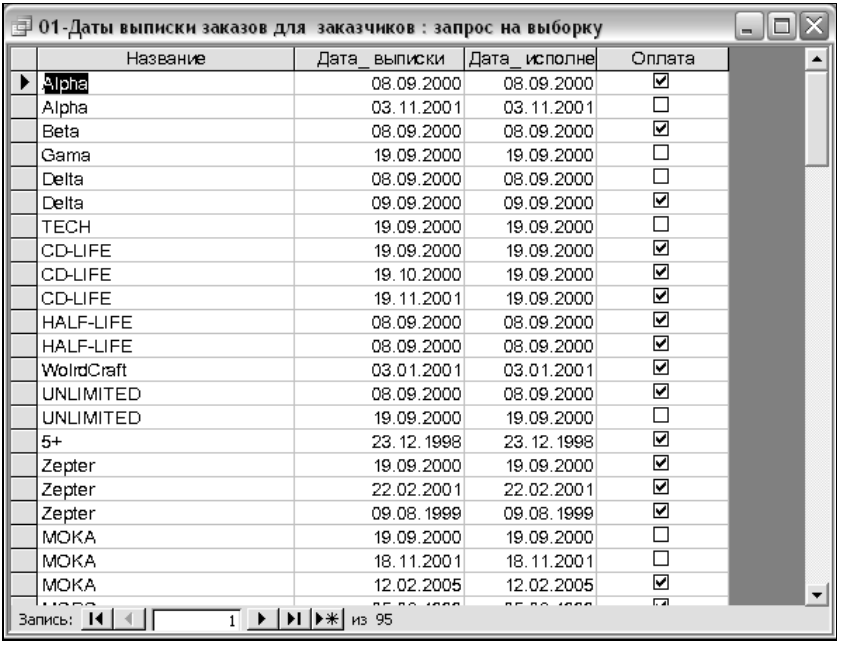


Рис. 6.43. Результаты запроса на выборку
с внутренним соединением по одному полю

8. Сохранить инструкцию запроса.

Замечание

Так как последовательность действий с диалоговым окном **Конструктор запроса** аналогична рассмотренному примеру, рекомендации, касающиеся последовательных действий при заполнении запроса, будут ограничиваться только необходимыми замеча-

ниями. Все таблицы, помещаемые в область таблиц запроса, и требования в области бланка запроса будут приводиться на соответствующих рисунках.

Пример. Получить список товаров по накладным с заказанным количеством и ценой (запрос на выборку с косвенными связями).

Решение.

Решение примера показано на рис. 6.44.

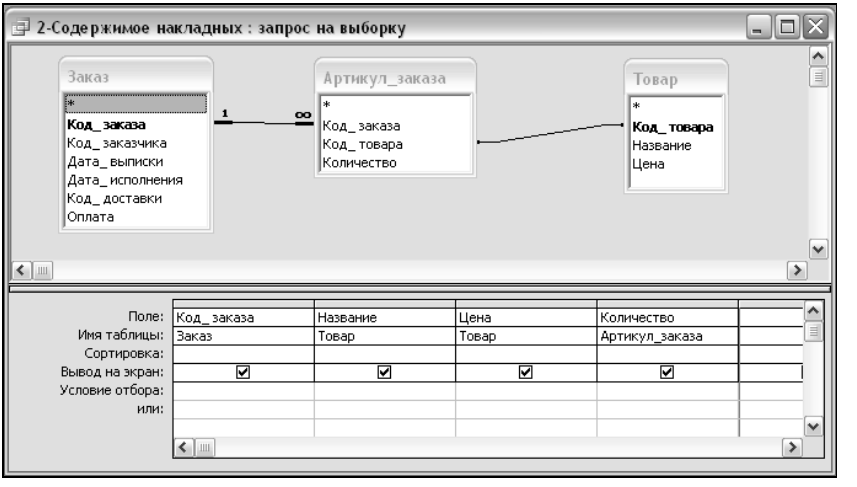


Рис. 6.44. Запрос на выборку с косвенными связями

В запрос включаются все таблицы, служащие звеном в цепочке соединений. В бланк запроса помещаются только нужные поля из всей этой связанной схемы.

Пример. Получить список заказчиков, у которых физический адрес и адрес доставки совпадают (внутреннее соединение по нескольким полям).

Решение.

Решение примера показано на рис. 6.45.

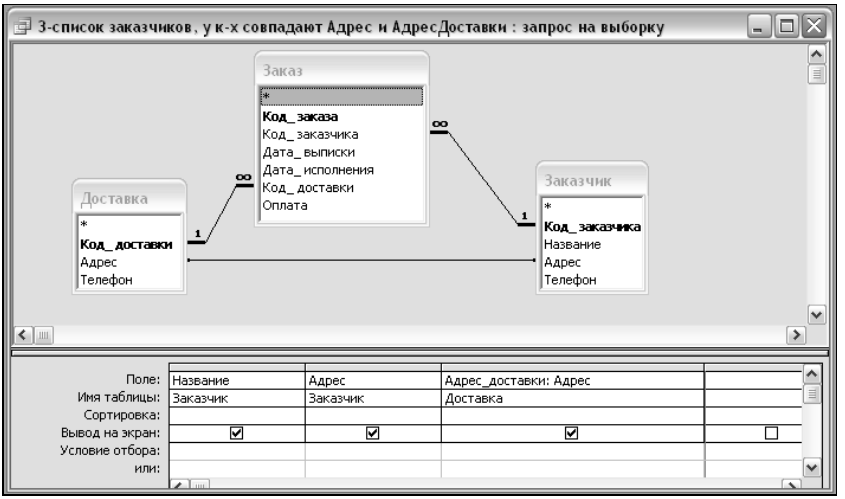


Рис. 6.45. Внутреннее соединение по нескольким полям

В данном примере необходимо провести выборку из таблиц, у которых возможно совпадение значений в ячейках, и вывести эти совпадения в виде результата. Для этого требуется создать внутреннее соединение между таблицами по интересующим полям ("перетаскиваем" одно поле, допускающее совпадение, на другое). При этом получается линия с точками по обеим сторонам, означающая, что соединение выполнено между полями, связь которых в схеме данных не задана, их имена не совпадают, и они не являются первичными ключами.

Для запрета вывода одинаковых строк необходимо поставить **Да** в строке **Уникальные значения** в диалоговом окне **Свойства запроса**.

Пример. Получить список заказчиков с номерами заказов, датами выписки и исполнения, оплатой. Добавить в запрос па-

раметр, требующий при выполнении запроса указать название заказчика (левое внешнее соединение с параметром).

Решение.

Решение примера показано на рис. 6.46.

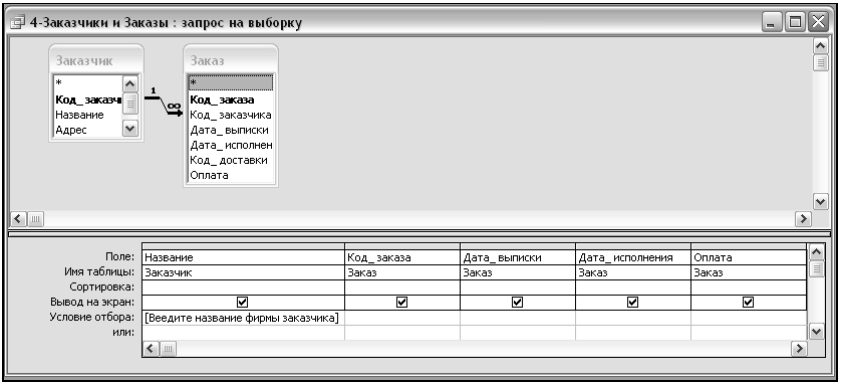


Рис. 6.46. Левое внешнее соединение

Внешние соединения позволяют вывести данные всех записей таблицы, участвующей в соединении независимо от того, имеются ли соответствующие им записи в связанной таблице. Внешние соединения бывают левыми или правыми. Запрос, в котором участвуют таблицы с левым внешним соединением, выводит все записи родительской таблицы с уникальным значением первичного ключа вне зависимости от того, имеются ли соответствующие им записи в дочерней таблице. И наоборот, запрос, в котором участвуют таблицы с правым внешним соединением, выводит все записи дочерней таблицы.

Для создания такого запроса следует изменить либо добавить необходимый вид связи между таблицами, помещенными в запрос:

1. Установить "перетаскиванием" связь между совпадающими полями.

2. Выбрать вид связи, делая двойной щелчок мышью по линии связи.
3. В появившемся диалоговом окне **Параметры объединения** (рис. 6.47) выбираем один из следующих вариантов: 1 — обычное внутреннее соединение; 2 — левое внешнее соединение; 3 — правое внешнее соединение.

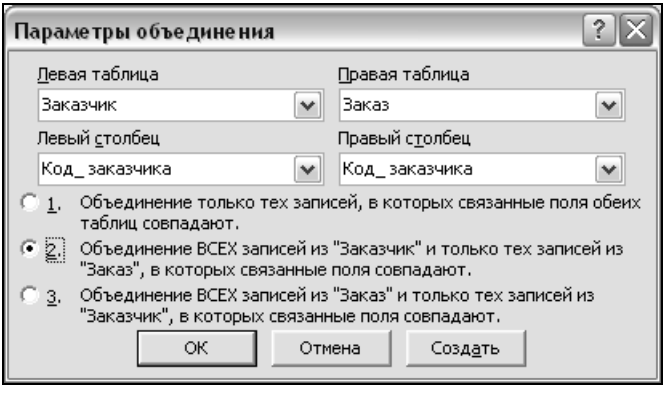


Рис. 6.47. Диалоговое окно **Параметры объединения** при изменении связи в запросе

4. Для добавления параметра к запросу в поле **Условие отбора** бланка запроса необходимо ввести текст сообщения в прямоугольных скобках

```
[Введите название фирмы заказчика]
```

Пример. Получить заказы, у которых совпадают даты выписки и исполнения (рекурсивное соединение).

Решение.

Решение примера показано на рис. 6.48.

Для получения рекурсивного соединения в запрос необходимо добавить копию таблицы, а затем создать соединение между полями `Дата_выписки` и `Дата_исполнения`.

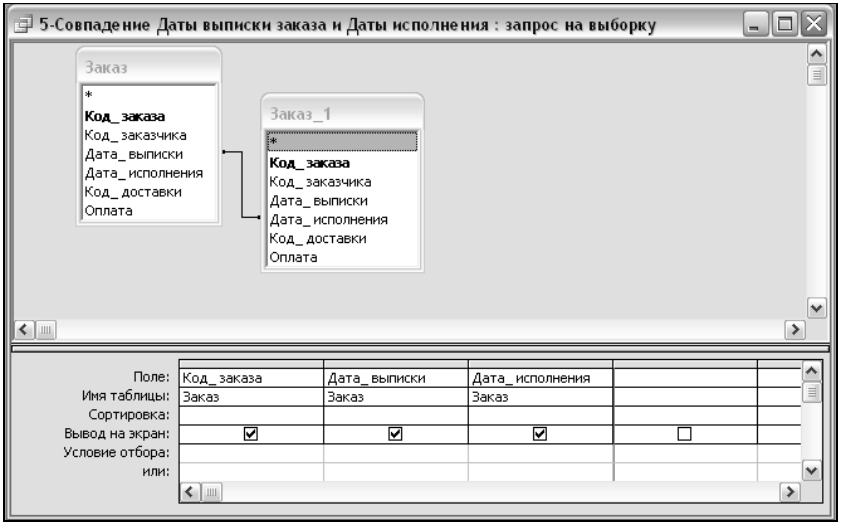


Рис. 6.48. Рекурсивное соединение

Для запрета вывода одинаковых строк необходимо поставить **Да** в строке **Уникальные значения** в диалоговом окне **Свойства запроса**.

Пример. Получить список фирм-заказчиков, которые имеют разный физический адрес и адрес доставки (соединение по отношению).

Решение.

Решение примера показано на рис. 6.49.

Связь, эквивалентную соединению по отношению, можно задать с помощью условия отбора, которое указывается для одного из двух полей, участвующих в соединении.

Для запрета вывода одинаковых строк необходимо поставить **Да** в строке **Уникальные значения** в диалоговом окне **Свойства запроса**.

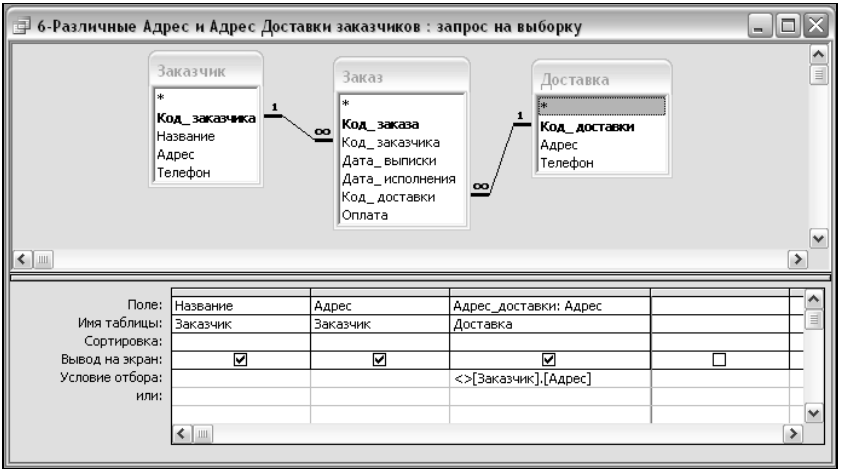


Рис. 6.49. Соединение по отношению

Результаты действия данного запроса приведены на рис. 6.50.

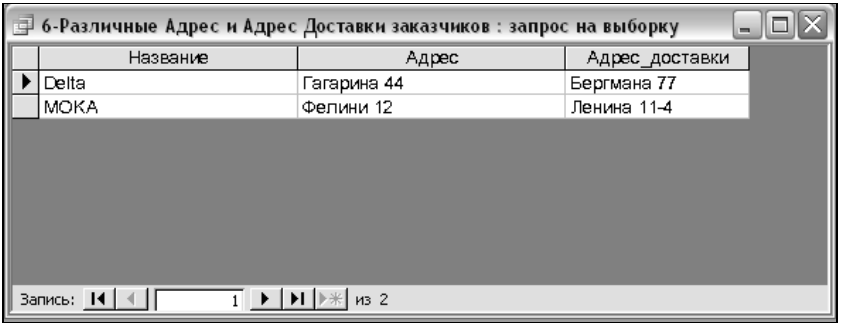


Рис. 6.50. Результат действия запроса, включающего соединение по отношению

Пример. Получить стоимость отдельного товара в заказах (запрос с вычисляемым полем).

Решение.

Решение примера показано на рис. 6.51.

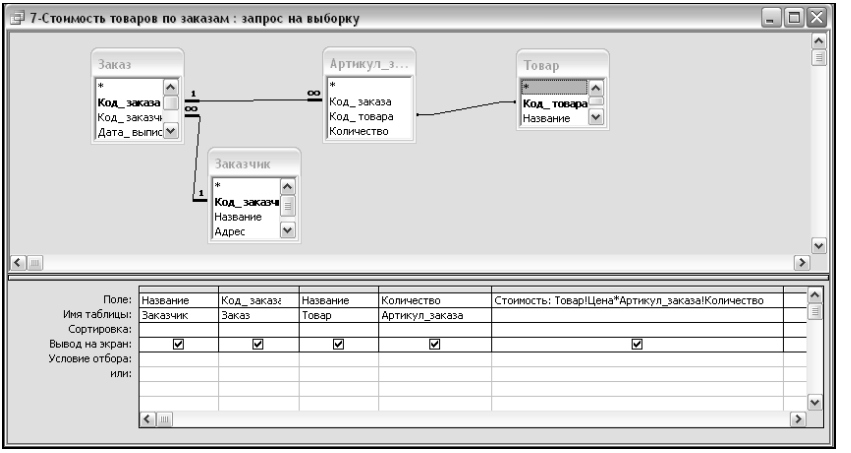


Рис. 6.51. Запрос с вычисляемым полем

В поле, которое будет вычисляемым, вводится название `Стоимость:` (двоеточие после названия обязательно!) и затем нажимается кнопка  (построить) на панели инструментов **Конструктор запросов**.

В диалоговом окне **Построитель выражения** создается соответствующая формула, окончательный вид которой для вычисляемого поля следующий:

```
Стоимость: Товар!Цена*Артикул_заказа!Количество
```

Пример. Получить суммарную стоимость конкретных товаров, оформленных в заказах (запрос с вычисляемым полем и групповой операцией).

Решение.

Решение примера показано на рис. 6.52.

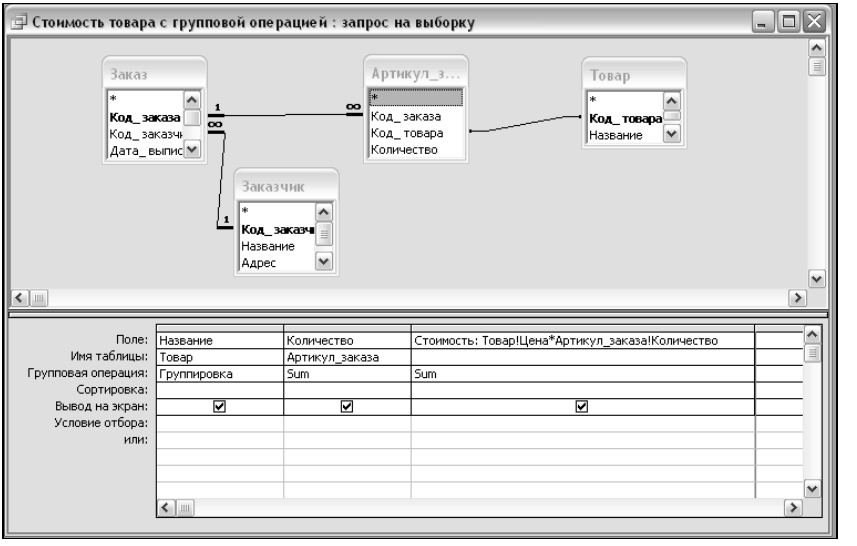


Рис. 6.52. Запрос с вычисляемым полем и групповой операцией

Для решения примера необходимо выполнить следующие действия:

1. В бланк запроса добавить необходимые поля: `Название` (табл. 6.7) и `Количество` (табл. 6.11).
2. Сформировать вычисляемое поле стоимость:

   `Стоимость: Товар!Цена*Артикул_заказа!Количество`
3. Воспользоваться командой **Групповые операции** в меню **Вид** либо кнопкой Σ на панели инструментов **Конструктор запросов** для добавления поля **Групповая операция** в бланк запроса.
4. Установить для поля **Групповая операция** в бланке запроса следующие значения для соответствующих полей:
   - `Название` — **Группировка**;
   - `Количество` — **Sum**;
   - `Стоимость` — **Sum**.

Пример. Определить суммарную стоимость товара: "шкаф трехстворчатый", "вешалка для одежды" и "доска объявлений", дата выписки которых по накладным начинается с 1 января 2001 г., рассчитать также налоговые отчисления за конкретный товар и общую сумму оплаты за товар (стоимость и налоговые отчисления). Правило для расчета налоговых отчислений: отчисления производятся со стоимости каждого проданного товара и представляют итоговое значение в денежном формате. На сумму отчислений налагаются условия: стоимость товара менее 600 000 р. — удерживается 10% суммы, стоимость товара, более 600 000 р. — 50%.

Решение.

Решение примера показано на рис. 6.53.

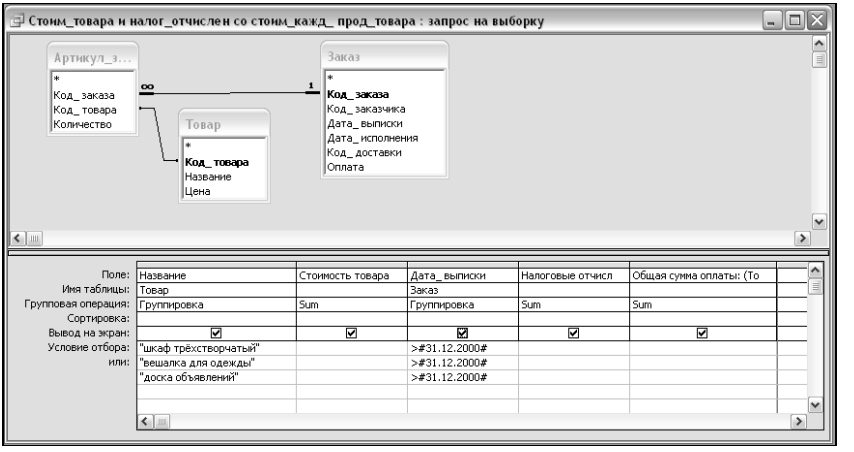


Рис. 6.53. Запрос на использование вычисляемых полей, группировок и условий отбора

Для решения примера необходимо выполнить следующие действия:

1. Добавить в бланк запроса необходимые поля (`Название` из табл. 6.7 и `Дата_выписки` из табл. 6.10).

2. Сформировать вычисляемые поля:

   - стоимость товара

     ```
     Стоимость товара: Товар!Цена*Артикул_заказа!Количество
     ```

   - налоговые отчисления

     ```
     Налоговые отчисления:
     FormatCurrency(IIf

     (Товар!Цена*Артикул_заказа!Количество<600000;
                (0,1*Товар!Цена*Артикул_заказа!Количество);
                (0,5*Товар!Цена*Артикул_заказа!Количество)))
     ```

   - общая сумма оплаты

     ```
     Общая сумма оплаты:
     (Товар!Цена*Артикул_заказа!Количество)+
     FormatCurrency(IIf

     (Товар!Цена*Артикул_заказа!Количество<600000;
                (0,1*Товар!Цена*Артикул_заказа!Количество);
                (0,5*Товар!Цена*Артикул_заказа!Количество)))
     ```

3. Добавить необходимые условия отбора в поле `Название` и в поле `Дата_выписки` в соответствии с рис. 6.53.

4. Воспользоваться командой **Групповые операции** в меню **Вид** либо кнопкой  на панели инструментов **Конструктор запросов** для добавления поля **Групповая операция** в бланк запроса.

5. Установить в поле **Групповая операция** в бланке запроса следующие значения для соответствующих полей:

   - `Название` — **Группировка**;
   - `Стоимость товара` — **Sum**;
   - `Дата_выписки` — **Группировка**;
   - `Налоговые отчисления` — **Sum**;
   - `Общая сумма оплаты` — **Sum**.

Запросы на изменение (модифицирующие запросы)

Пример. Создать таблицу `Адрес_доставки`, включающую информацию о клиентах, имеющих разные физический адрес и адрес доставки (запрос на создание таблицы).

Решение.

Решение примера показано на рис. 6.54.

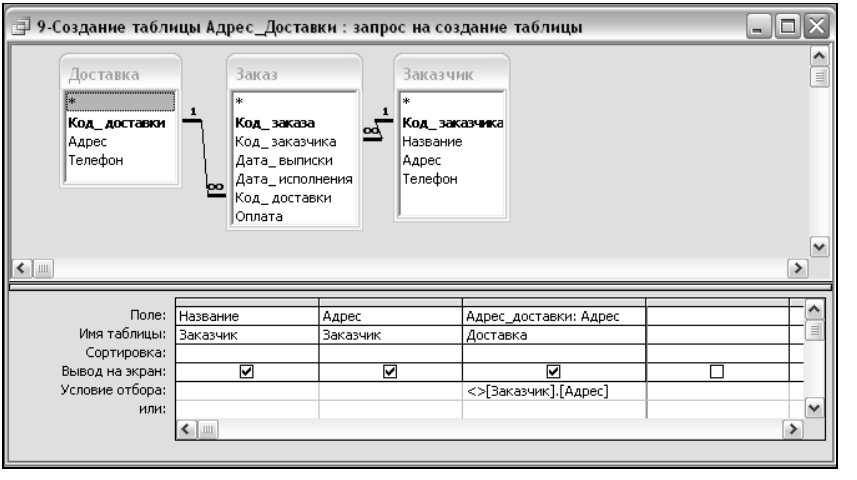


Рис. 6.54. Запрос на создание таблицы

Запросы на создание таблицы позволяют создавать таблицы из результатов какого-либо запроса. Как правило, этот тип запросов используется для экспорта данных в другие приложения либо в другую БД.

Построение запроса включает несколько этапов:

1. В диалоговое окно **Конструктор запроса** добавить табл. 6.8—6.10 и определить условия выполнения в бланке запроса (рис. 6.54):

   - воспользоваться командой **Создание таблицы** в меню **Запрос** либо кнопкой  **(Тип запроса)** на панели инст-

рументов **Конструктор запросов** и в появившееся диалоговое окно **Создание таблицы** ввести имя новой таблицы — `Адрес_доставки`;

- сохранить запрос в БД (все запросы на создание таблиц помечаются в окне БД значком).

2. После запуска запроса на создание таблицы в БД создается новая таблица, которую можно просмотреть либо отредактировать, перейдя к объекту таблицы в окне БД.

Пример. Добавить в таблицу `Адрес_доставки` записи о совпадающих адресах заказчиков (запрос на добавление).

Решение.

Решение примера показано на рис. 6.55.

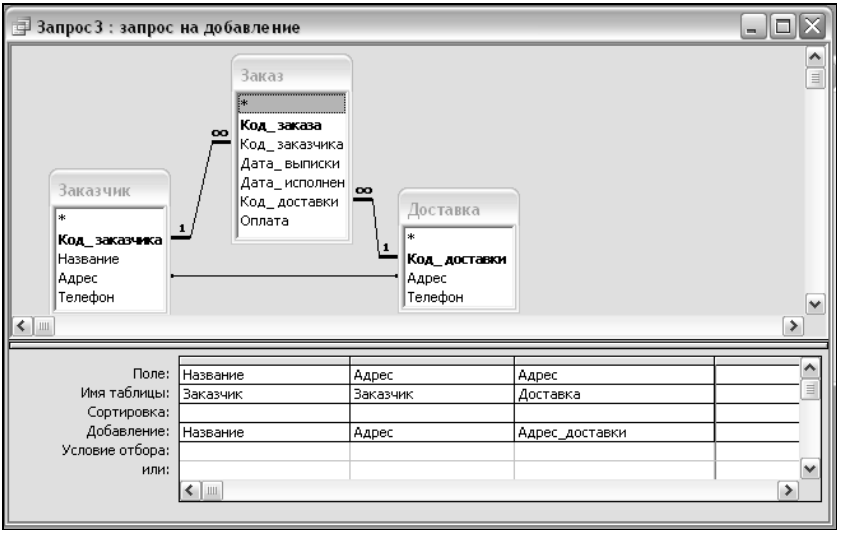


Рис. 6.55. Запрос на добавление

Для решения примера необходимо выполнить следующие действия:

1. Подготовить запрос на выборку данных с учетом того, что они будут добавлены в некоторую таблицу.
2. Выбрать команду **Добавление** в меню **Запрос** либо воспользоваться кнопкой  на панели инструментов **Конструктор запросов**.
3. В открывшемся диалоговом окне **Добавление** нужно указать таблицу, к которой будут добавлены записи, — `Адрес_доставки`. Следует помнить, что для добавления данных необходимо, чтобы имена полей запроса и полей таблицы, в которую будут вноситься значения, были одинаковы и имели одинаковый тип данных.
4. Сохранить запрос в БД (все запросы на добавление помечаются в окне БД значком ).
5. После запуска запроса на добавление данных к таблице в таблицу `Адрес_доставки` добавляются новые данные, которые можно просмотреть либо отредактировать, перейдя к таблице `Адрес_доставки` в окне БД.

Пример. Удалить из таблицы `Адрес_доставки` записи, которые удовлетворяют следующим критериям: название фирмы заказчика содержит первую букву "М" либо букву "о" в середине названия и в физическом адресе фирмы (запрос на удаление).

Решение.

Решение примера показано на рис. 6.56.

Запросы на удаление позволяют удалить из таблицы записи, соответствующие некоторым критериям. Перед удалением записей рекомендуется вывести их, используя запрос на выборку, а затем этот запрос преобразовать в запрос на удаление.

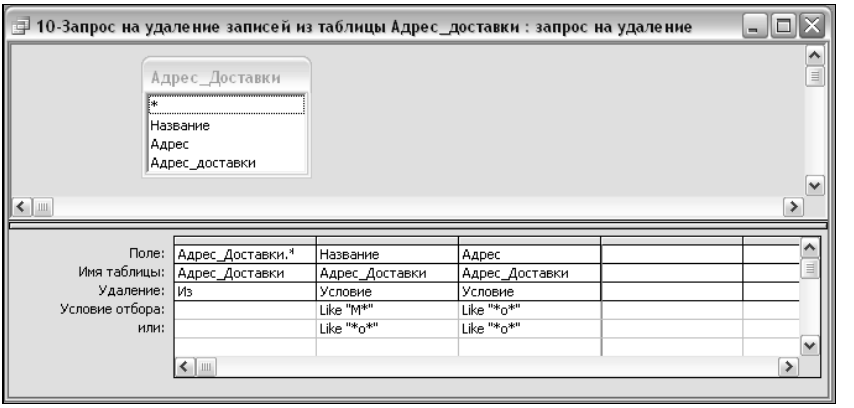


Рис. 6.56. Запрос на удаление

При создании запроса на удаление рекомендуется придерживаться следующих действий:

1. Выбрать создание запроса в режиме конструктора и добавить таблицу, из которой будут удаляться записи (в нашем случае — `Адрес_Доставки`).
2. Выбрать команду **Удаление** в меню **Запрос** либо воспользоваться кнопкой  на панели инструментов **Конструктор запросов**.
3. При переходе к режиму на удаление в бланке запроса исчезают строки **Сортировка** и **Вывод на экран**, а появляется строка **Удаление**.
4. Заполнить бланк запроса (рис. 6.56).
5. Сохранить запрос в БД (все запросы на удаление помечаются в окне БД значком ).
6. После запуска запроса на удаление данных в таблице записи в таблице `Адрес_доставки` удаляются навсегда.
7. Для просмотра результата следует перейти к таблице `Адрес_доставки` в окне БД.

При выполнении запросов на удаление следует помнить, что удаление записей в родительской таблице (со стороны "один"), которым соответствуют записи в дочерней таблице (со стороны "многие"), нарушает условие целостности данных, поэтому связанные записи в таблице со стороны "многие" становятся "висячими". Для уничтожения данных необходимо использовать каскадное удаление: сначала требуется удалить записи из таблицы с отношением "многие", а затем из таблицы с отношением "один".

Пример. Обновить данные в таблице `Адрес_доставки` в соответствии со следующими критериями: для фирмы "МОКА" физический адрес изменился на "Кленовая, 14", адрес доставки — "Врублевского, 12" (запрос на обновление).

Решение.

Решение примера показано на рис. 6.57.

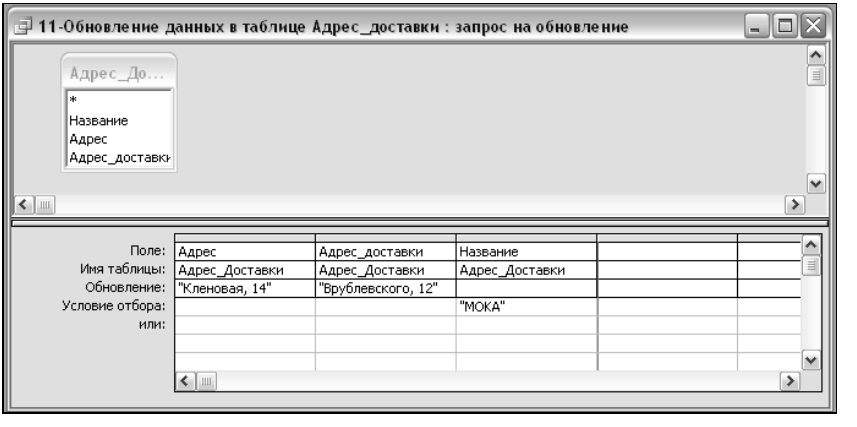


Рис. 6.57. Запрос на обновление

Запрос на обновление изменяет значения существующих полей таблицы в соответствии с критериями запроса.

При создании запроса на обновление рекомендуется придерживаться следующей последовательности действий:

1. Выбрать создание запроса в режиме конструктора и добавить таблицу `Адрес_Доставки`.
2. Выбрать команду **Обновление** в меню **Запрос** либо воспользоваться кнопкой  (тип запроса) на панели инструментов **Конструктор запросов**.
3. При переходе к режиму на обновление в бланке запроса исчезают строки **Сортировка** и **Вывод на экран**, а появляется строка **Обновление**.
4. Заполнить бланк запроса с учетом изменений и условий отбора (рис. 6.57).
5. Сохранить запрос в БД (все запросы на обновление помечаются в окне БД значком ).
6. После запуска запроса на обновление данных в таблице записи в таблице `Адрес_доставки` обновляются в соответствии с требованиями запроса.
7. Для просмотра результата следует перейти к таблице `Адрес_доставки` в окне БД.

Замечание

Все модифицирующие запросы при запуске их на выполнение требуют подтверждения действий (рис. 6.58).

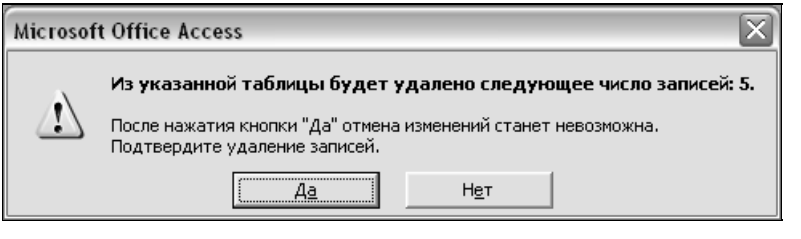


Рис. 6.58. Окно, требующее подтверждения запроса на изменение

Перед запуском запросов на модификацию рекомендуется создать резервную копию таблицы во избежание потери необходимых данных.

Перекрестные запросы

Перекрестные запросы (или кросс-таблицы) представляют собой двумерные таблицы, в которых, как правило, показана некоторая вычисляемая статистическая информация. Перекрестный запрос всегда имеет три элемента:

- *заголовки строк* — до трех значений полей (можно использовать и различные групповые операции);
- *заголовки столбцов* — значение одного поля;
- *значение* — значение одного поля либо вычисляемое значение, которое располагается на пересечении строк и столбцов итоговой кросс-таблицы.

Перекрестные запросы также могут содержать:

- условия отбора;
- групповые операции;
- сортировки;
- вычисляемые поля (как в качестве значений перекрестной таблицы, так и в качестве заголовков столбцов).

Перекрестные запросы позволяют:

- получить большой объем данных в компактном виде;
- формировать графики и диаграммы в MS Access;
- просматривать уровни детализации.

Замечание

При использовании перекрестных запросов нельзя сортировать таблицу результатов по значениям, содержащимся в столбцах.

Пример. Получить стоимость заказов с учетом товаров, приобретенных после 26 июня 2001 г. (перекрестный запрос).

Решение.

Решение примера показано на рис. 6.59.

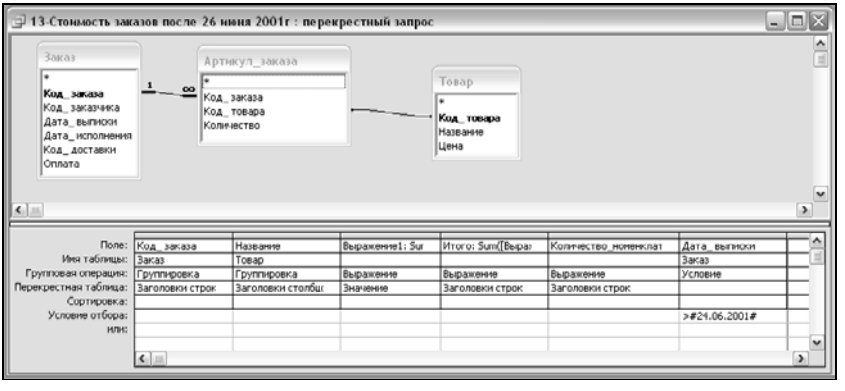
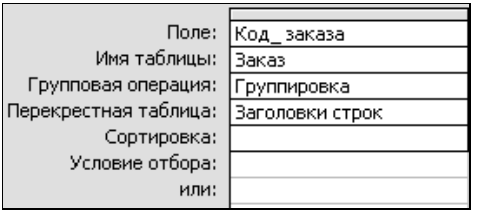


Рис. 6.59. Перекрестный запрос

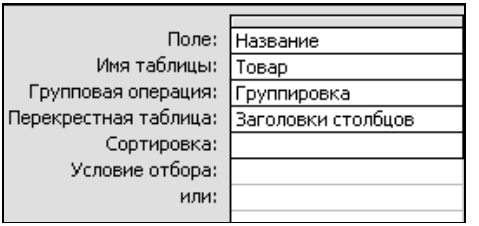
При создании запроса рекомендуется придерживаться следующей последовательности действий:

1. Выбрать создание запроса в режиме конструктора и добавить табл. 6.7, 6.10 и `Артикул_заказа`.
2. Выбрать команду **Перекрестный** в меню **Запрос** либо воспользоваться кнопкой  на панели инструментов **Конструктор запросов**.
3. При переходе к режиму создания перекрестного запроса в бланк запроса добавляется строка *перекрестная таблица.*
4. Заполнить бланк запроса следующим образом:
   - заполнение поля `Код_заказа` показано на рис. 6.60;
   - заполнение поля `Название` показано на рис. 6.61;



Поле:	Код_заказа
Имя таблицы:	Заказ
Групповая операция:	Группировка
Перекрестная таблица:	Заголовки строк
Сортировка:	
Условие отбора:	
или:	

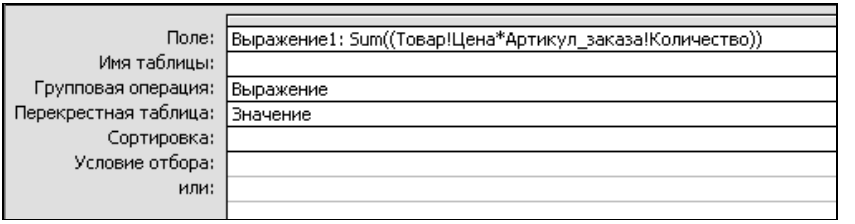
Рис. 6.60. Заполнение бланка перекрестного запроса для поля `Код_заказа`



Поле:	Название
Имя таблицы:	Товар
Групповая операция:	Группировка
Перекрестная таблица:	Заголовки столбцов
Сортировка:	
Условие отбора:	
или:	

Рис. 6.61. Заполнение бланка перекрестного запроса для поля `Название`

- заполнение поля вычисляемого значения стоимости товара, расположенного на пересечении строк и столбцов, показано на рис. 6.62;



Поле:	Выражение1: Sum((Товар!Цена*Артикул_заказа!Количество))
Имя таблицы:	
Групповая операция:	Выражение
Перекрестная таблица:	Значение
Сортировка:	
Условие отбора:	
или:	

Рис. 6.62. Заполнение бланка перекрестного запроса для поля вычисляемого значения стоимости товара

- заполнение поля для итогового значения стоимости заказа показано на рис. 6.63;

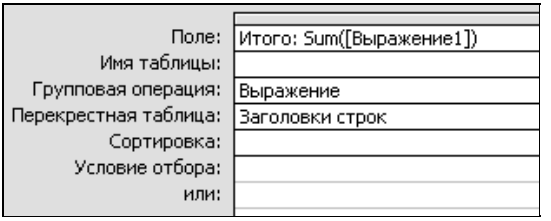


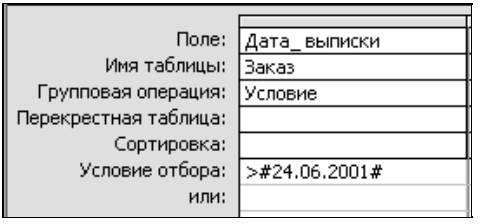
Рис. 6.63. Заполнение бланка перекрестного запроса для итогового значения стоимости заказа

- заполнение поля для подсчета количества номенклатуры заказа показано на рис. 6.64;

Рис. 6.64. Заполнение бланка перекрестного запроса для подсчета количества номенклатуры заказа

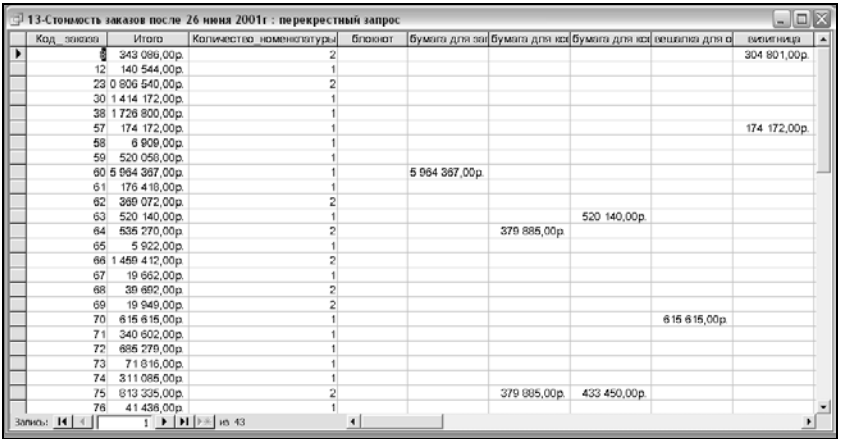
- заполнение поля `Дата_выписки` показано на рис. 6.65.

5. Сохранить запрос в БД (все перекрестные запросы помечаются в окне БД значком).
6. После запуска запроса на выполнение результаты выводятся в виде двумерной таблицы (рис. 6.66).



Поле:	Дата_выписки
Имя таблицы:	Заказ
Групповая операция:	Условие
Перекрестная таблица:	
Сортировка:	
Условие отбора:	>#24.06.2001#
или:	

Рис. 6.65. Заполнение бланка перекрестного запроса для поля `Дата_выписки`



13-Стоимость заказов после 26 июня 2001г : перекрестный запрос

Код_заказа	Итого	Количество_номенклатуры	блокнот	бумага для зам	бумага для кан	бумага для кан	ежедневка для о	визитница
8	343 086,00р.	2						304 801,00р.
12	140 544,00р.	1						
23	0 806 540,00р.	2						
30	1 414 172,00р.	1						
38	1 726 800,00р.	1						
57	174 172,00р.	1						174 172,00р.
58	6 909,00р.	1						
59	520 056,00р.	1						
60	5 964 367,00р.	1		5 964 367,00р.				
61	176 416,00р.	1						
62	369 072,00р.	2						
63	520 140,00р.	1				520 140,00р.		
64	535 270,00р.	2			379 885,00р.			
65	5 922,00р.	1						
66	1 456 412,00р.	2						
67	19 662,00р.	1						
68	39 692,00р.	2						
69	19 949,00р.	2						
70	615 615,00р.	1					615 615,00р.	
71	340 602,00р.	1						
72	685 279,00р.	1						
73	71 616,00р.	1						
74	311 085,00р.	1						
75	813 335,00р.	2			379 885,00р.	433 450,00р.		
76	41 436,00р.	1						

Запись: 1 из 43

Рис. 6.66. Результаты выполнения перекрестного запроса

На основе перекрестного запроса удобно создавать сводные таблицы и диаграммы в MS Access (рис. 6.67).

Замечание

Все рассмотренные ранее типы запросов могут включаться в качестве источника данных (подзапроса) в другие запросы.

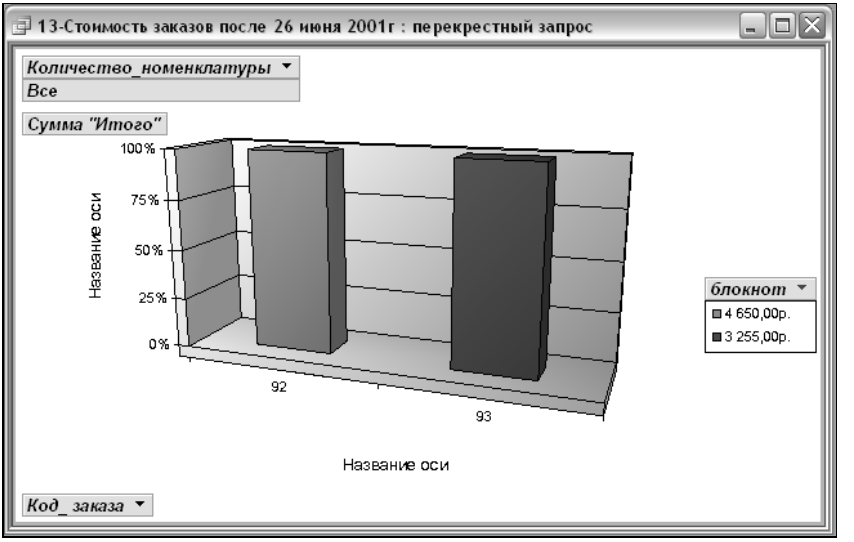


Рис. 6.67. Диаграмма, созданная на основе перекрестного запроса

Создание форм и отчетов. Использование макросов

Создание форм и отчетов в MS Access, включая использование макросов, позволяет придать приложению, созданному на основе базы данных, законченный вид: оконный интерфейс, автоматизация действий пользователя, распечатка необходимых документов, полученных на основе хранимых данных.

Создание форм

Формы в MS Access используются для поддержки следующих задач:

- ввод, редактирование и просмотр информации, находящейся в таблицах;

- отображение результатов запросов;
- организация экранного окна поиска необходимой информации по БД (с использованием различных условий и макросов);
- распечатка данных в организованном виде;
- сохранение в виде отчета с последующей модификацией;
- создание пользовательских экранных форм, облегчающих работу с ДБ в целом.

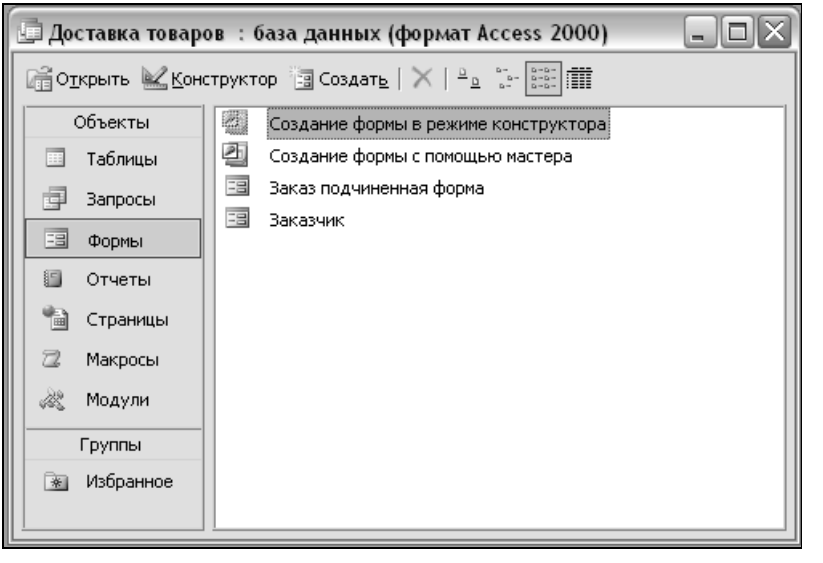


Рис. 6.68. Окно БД
с выбранными объектами формы

Создание новой формы в MS Access можно осуществить с использованием кнопки 

в окне БД при переходе к объекту **Формы** (рис. 6.68). Однако существуют и другие способы создания форм.

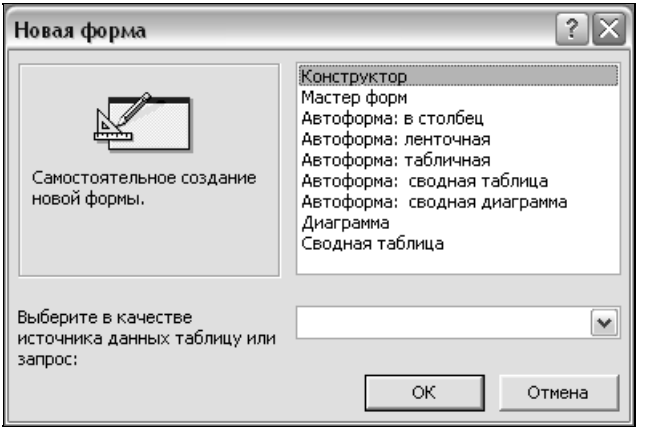


Рис. 6.69. Диалоговое окно **Новая форма**

Имеются следующие возможности создания форм (рис. 6.69):

- **Конструктор** — позволяет создать новую форму самостоятельно с использованием окна конструктора форм (переход в этот режим можно также осуществить с помощью команды **Создание формы в режиме конструктора**, отображаемой в окне БД, см. рис. 6.68);
- **Мастер форм** — автоматическое создание формы одного из трех стандартных типов (в столбец, ленточную или табличную) на основе выбранных полей (переход в этот режим можно также осуществить с помощью команды **Создание формы с помощью мастера**, отображаемой в окне БД, см. рис. 6.68);
- **Автоформа: в столбец** — автоматическое создание формы с полями в один столбец;
- **Автоформа: ленточная** — автоматическое создание ленточной формы;
- **Автоформа: табличная** — автоматическое создание табличной формы;

- **Автоформа: сводная таблица** — автоматическое создание формы в режиме сводной таблицы;
- **Автоформа: сводная диаграмма** — автоматическое создание формы в режиме сводной диаграммы;
- **Диаграмма** — создание формы с диаграммой;
- **Сводная таблица** — создание формы со сводной таблицей.

Наиболее широкие возможности по созданию форм предлагает окно конструктора форм (рис. 6.70).

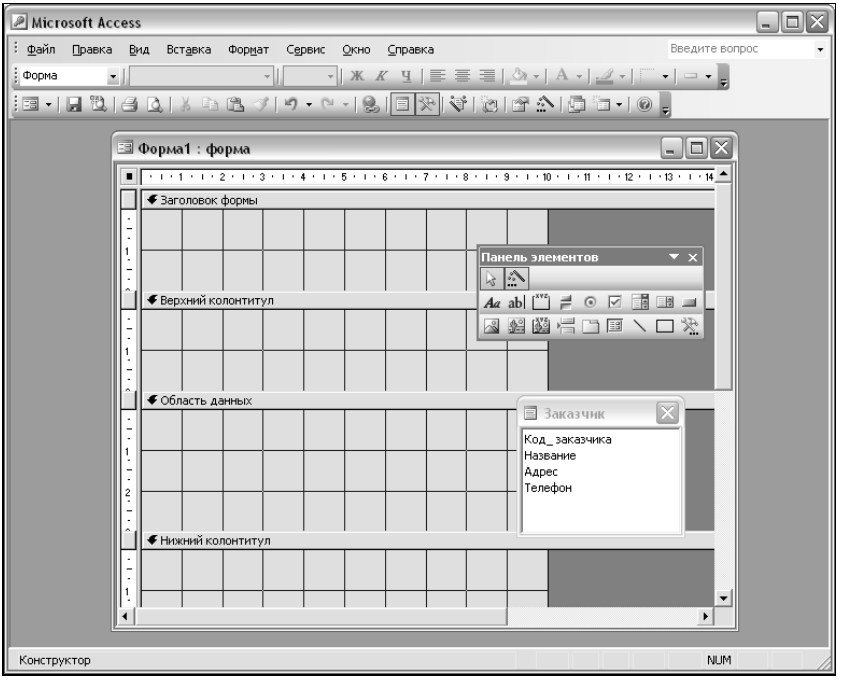


Рис. 6.70. Окно конструктора формы

Следует заметить, что перед тем, как создавать форму, необходимо определиться с таблицей (либо запросом), для кото-

рой она создается. Если форма использует данные нескольких таблиц, то рекомендуется предварительно создать запрос, который включает необходимые данные. Определить либо изменить источник данных для формы можно с помощью диалогового окна **Форма** (рис. 6.71), которое может быть открыто с помощью команды **Свойства** в меню **Вид** либо кнопки  на панели инструментов **Конструктор форм**. Для получения свойств всей формы в целом необходимо щелкнуть мышью на заголовке окна конструктора форм.

Форма и отчет представляют собой тип объекта, обладающий структурным делением на области, в которые можно поместить разнообразные элементы.

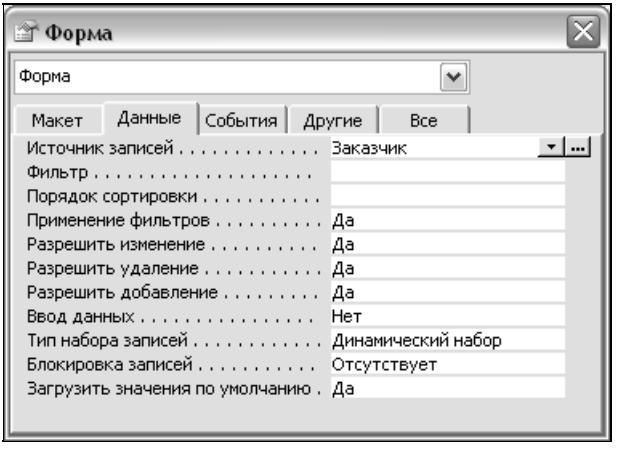


Рис. 6.71. Диалоговое окно **Форма**

Создание и редактирование формы происходит непосредственно в окне конструктора форм (рис. 6.70), в котором могут быть выделены следующие области:

- *область данных* — область вычисляемых полей, предназначенная для размещения полей из таблицы или запроса;

- *заголовок формы* (команда **Заголовок/Примечание формы** в меню **Вид**) — используется для отображения заголовка и даты;
- *примечание формы* (команда **Заголовок/Примечание формы** в меню **Вид**) — используется для помещения инструкций, связанных с общей информацией по заполнению формы, а также некоторой общей информацией, связанной с вычисляемыми полями;
- *верхний и нижний колонтитулы* (команда **Колонтитулы** в меню **Вид**) — предназначены для помещения на каждой странице формы некоторой постоянной информации.

Кроме того, с использованием команд категории **Вид** в окно конструктора форм (рис. 6.70) можно добавить сетку и линейку (для удобства конструирования формы), отобразить список полей таблицы или запроса (источник данных для формы), вывести панель элементов (для размещения в областях формы различных объектов).

Рис. 6.72. Панель элементов конструктора форма

Для редактирования всех элементов окна и объектов, которые помещаются в форму, используются стандартные приемы и возможности команд меню и панелей инструментов.

Следует отметить возможности элементов управления, которые широко используются при создании форм и отчетов. Элементы управления выводятся на экран с использованием команды **Панель элементов** в меню **Вид** либо с использованием кнопки на панели инструментов **Конструктор форм**. Основные элементы управления **Панели элементов** (рис. 6.72) приведены в табл. 6.17.

Таблица 6.17. *Основные элементы управления*

Кнопка	Описание элемента управления
Основные элементы управления	
 (надпись)	Служит для ввода и отображения текста
 (поле)	Позволяет размещать на форме текстовую надпись и поле для ввода данных
 (группа переключателей)	Содержит несколько выключателей, переключателей и флажков
 (выключатель)	Связан с логическим полем и может находиться в двух состояниях: нажатом либо отжатом. Выключатели могут быть добавлены в группу
Расширенные элементы управления ввода данных	
 (переключатель)	Используется для установки переключателей в группе, в которой только один может быть активным
 (флажок)	Обычно используется для отображения логического типа данных. Флажки могут быть объединены в группу
 (поле со списком)	Представляет собой раскрывающийся список значений, в который можно вводить также и другие значения
 (список)	Список заданных значений для удобства ввода данных
 (кнопка)	Используется для вызова макроса, запуска программы на VBA или для выполнения какого-либо действия
 (вкладка)	Отображает несколько страниц в виде папки с вкладками

Таблица 6.17 *(продолжение)*

Кнопка	Описание элемента управления
Расширенные элементы управления ввода данных	
 (подчиненная форма/ отчет)	Позволяет отобразить еще одну форму (или отчет) в основном объекте
Элементы управления графикой	
 (рисунок)	Отображает на экране рисунок
 (свободная рамка объекта)	Содержит объект OLE, рисунок, диаграммы, звук, видео, не связанные с полем таблицы
 (присоединенная рамка объекта)	Содержит объект OLE или рисунок, связанные с полем таблицы
 (разрыв страницы)	Обычно используется в отчетах и создает разрыв страницы. Однако может быть помещен в многостраничную форму
 (линия)	Используется для разделения элементов управления либо взаимосвязанных данных
 (прямоугольник)	Используется для выделения либо разделения элементов формы
Дополнительные кнопки	
 (выбор объектов)	Позволяет выделить несколько объектов в группу и работать с ними как одним целым

Таблица 6.17 *(окончание)*

Кнопка	Описание элемента управления
Дополнительные кнопки	
 (мастер)	В MS Access имеется большое количество мастеров, позволяющих оптимизировать работу с различными элементами, помещаемыми на форму (либо отчет). Активизированная кнопка мастера свидетельствует о том, что при работе с конкретным объектом панели элементов запустится соответствующий мастер
 (другие элементы)	Данная кнопка позволяет осуществить вызов других объектов, которые могут быть размещены на форме

Существуют три основных типа элементов управления.

- *Присоединенные элементы управления* — элементы управления, которые связаны с полем таблицы. При вводе значения в присоединенный элемент управления поле таблицы в текущей записи автоматически обновляется.
- *Свободные элементы управления* — элементы управления, которые сохраняют введенную величину без обновления поля таблицы. Они используются для отображения текста и значений, которые должны быть переданы макросам (например, кнопка, рамка, надпись и т. д.). Свободные элементы управления предназначены для сохранения объектов, которые содержатся не в таблице или запросе, а в самой форме.
- *Вычисляемые элементы управления* — элементы управления, которые создаются на основе вычисляемых выражений (например, вычисляемые поля, которые можно сформировать в режиме конструктора форм).

Приемы работы с элементами управления, такие как помещение их в форму, выделение, изменение местоположения, размеров и т. д. аналогичны приемам работы с графическими объектами.

Все объекты формы (сама форма, области формы, элементы и т. д.) обладают различными свойствами, для получения доступа к которым необходимо:

- выделить необходимый объект формы;
- воспользоваться командой **Свойства** в меню **Вид** или контекстном меню либо кнопкой на панели инструментов **Конструктор форм**;
- задать необходимые свойства в окне свойств (рис. 6.73) выбранного объекта (окно свойств представляет собой набор различных категорий, зависящих от выбранного объекта и расположенных на соответствующих вкладках окна).

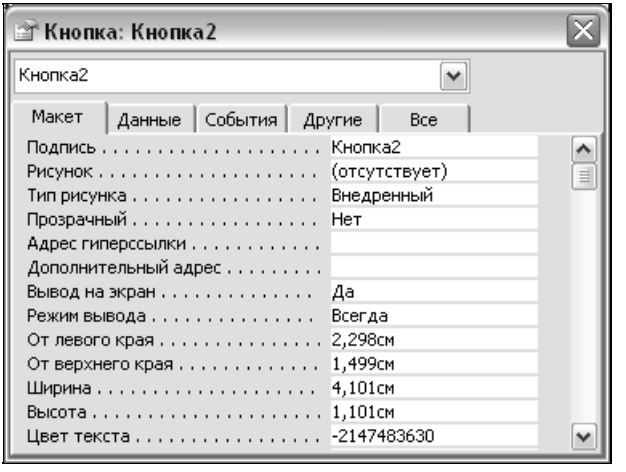


Рис. 6.73. Окно свойств элемента управления

Итак, для создания формы необходимо выполнить следующую последовательность действий:

1. Определить таблицы либо запросы, для которых будет создаваться форма.

2. Воспользоваться, например, командой **Создание формы** в режиме конструктора, отображаемой в окне БД (см. рис. 6.68).
3. В окне конструктора форм отобразить области, которые будут использоваться для создания макета формы.
4. Изменить (при необходимости) размер отображаемой формы, добавить линейку и сетку (для удобства).
5. Вывести список полей и элементов управления.
6. Добавить необходимые элементы в каждый раздел формы.
7. Применить форматирование к объектам и разделам формы, задать свойства объектов, разделов и формы.
8. Просмотреть готовую форму (команда **Режим формы** в меню **Вид** либо кнопка  на панели инструментов) **Конструктор форм**.
9. Внести (при необходимости) дополнительные изменения (команда **Конструктор** в меню **Вид**).
10. Сохранить форму.

Пример. Создать форму для ввода и отображения данных в таблице "Заказчик" с возможностью просмотра всех сделанных заказов.

Решение.

Результат решения примера показан на рис. 6.74.

Для решения примера необходимо:

1. Перейти в режим конструктора форм (выбрать, например, команду **Создание формы в режиме конструктора**, отображаемую в окне БД как показано на рис. 6.68).
2. Определить в свойствах формы таблицу `Заказчик` в качестве источника данных (рис. 6.75). Для этого щелчком правой кнопки мыши по кнопке  (см. рис. 6.70) вызвать системное меню формы и воспользоваться командой **Свойства**.

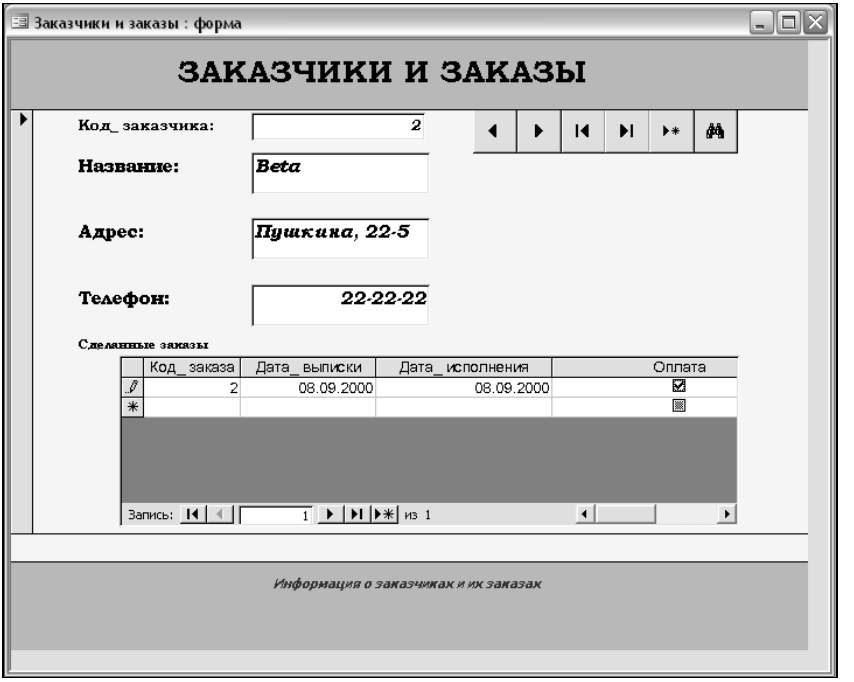


Рис. 6.74. Форма "Заказчики и заказы"

3. Вывести на экран область заголовков и примечаний формы, линейку и сетку, панель элементов и список полей (команды **Заголовок/Примечание формы**, **Линейка и Сетка**, **Панель элементов** и **Список полей** соответственно в меню **Вид**).

4. Добавить в область данных с помощью мыши необходимые элементы: поля данных из списка полей таблицы `Заказчик`, кнопки по работе с записями (объект **Кнопка** и объект **Подчиненная форма/отчет** на панели элементов). Наличие встроенных мастеров помогает осуществить необходимые операции по добавлению объектов.

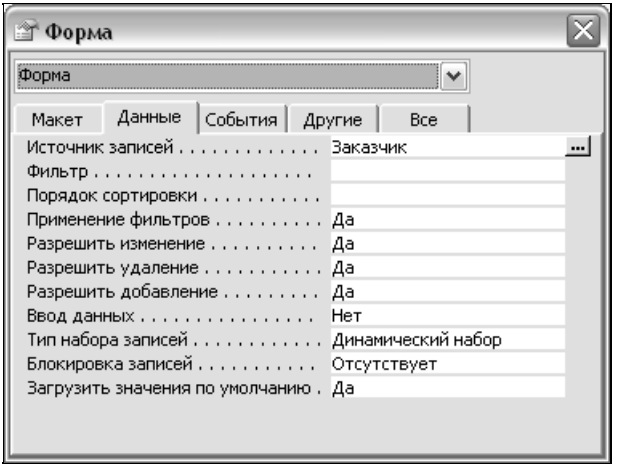


Рис. 6.75. Задание таблицы `Заказчик` в качестве источника данных для формы

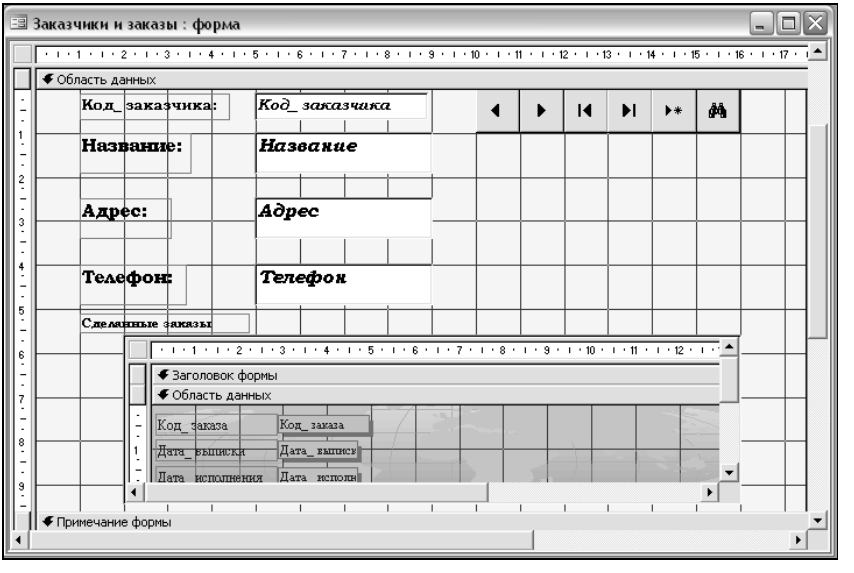


Рис. 6.76. Область данных в режиме конструктор форм

5. С помощью команд меню и панелей инструментов окна в режиме конструктора форм произвести необходимое редактирование и форматирование элементов, размещенных в области данных (рис. 6.76).

6. Добавить в область заголовков и примечаний формы объекты **Надпись** и ввести соответствующий текст. Произвести необходимое редактирование (рис. 6.77).

Рис. 6.77. Режим конструктора форм со всеми объектами для формы "Заказчики и заказы"

7. Вызвать окно свойств формы и на вкладке **Макет** установить для поля **Кнопки перехода** опцию **Нет**.

8. Просмотреть макет формы, используя команду **Режим формы** в меню **Вид** либо кнопку  на панели инструментов **Конструктор форм**.
9. Сохранить форму под именем "Заказчики и заказы".

Создание отчетов

Главное назначение отчетов — представление данных, находящихся в БД, в удобном в виде с целью их последующей распечатки либо использования в качестве визуально-сгруппированной информации.

Создание отчета предусматривает выполнение следующих основных этапов.

1. Определение макета отчета. На данном этапе необходимо определить цель отчета и составить общее представление о том, какого вида будет отчет. Эскиз отчета можно выполнить на бумаге либо с помощью окна конструктора отчетов MS Access. При этом следует учесть, что отчет, создаваемый в дальнейшем средствами MS Access, может включать:
   - иерархию в представлении данных (с помощью соответствующих группировок по необходимым полям);
   - формирование заголовков и примечаний, как для групп данных, так и для всего отчета;
   - использование необходимых вычислений, как для групп данных, так и для всего отчета;
   - формирование колонтитулов для всего отчета.
2. Сбор данных. На этом этапе необходимо определить, какая таблица либо запрос будет служить источником данных для создаваемого отчета. Если необходимая информация присутствует в различных таблицах и не созда-

но подходящего запроса, рекомендуется перед созданием отчета определиться с необходимыми полями и создать запрос, который будет служить источником данных для отчета.

3. Создание отчета, например, с помощью конструктора отчетов. По мере необходимости создаваемый отчет можно просматривать в экранном режиме и вносить соответствующие коррективы. По окончании работы над созданием отчета его рекомендуется сохранить.

4. Распечатка отчета либо использование его для других целей (например, подготовленный отчет можно отправить по электронной почте, выбрав в меню **Файл** пункт **Отправить**, команду **Сообщение (как вложение)**).

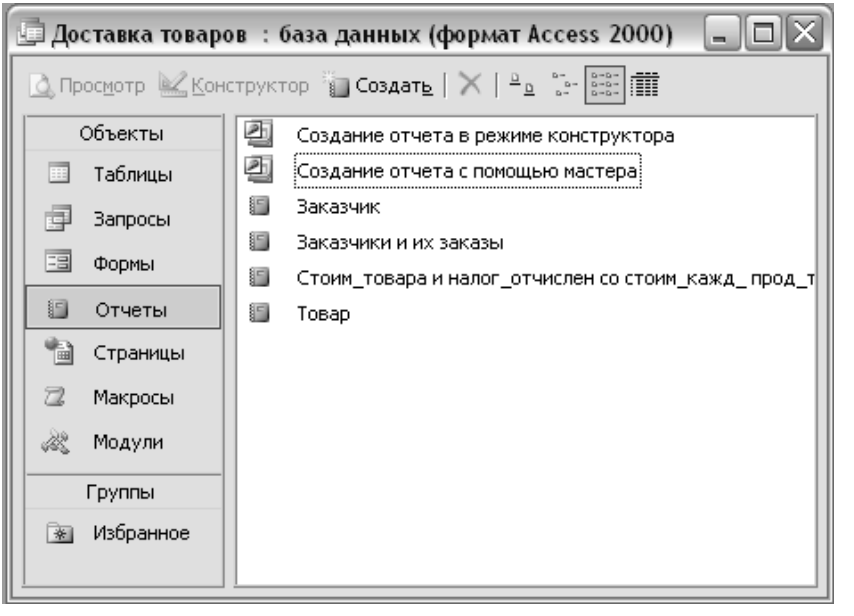


Рис. 6.78. Окно БД в режиме отчетов

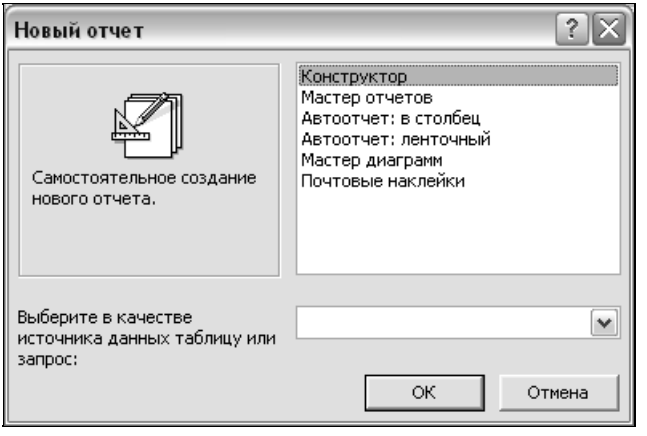


Рис. 6.79. Диалоговое окно **Новый отчет**

В MS Access есть много возможностей создания отчетов (выбрать кнопку Создать в окне БД), они показаны на рис. 6.78 и 6.79.

- **Конструктор** — позволяет самостоятельно проектировать и создавать отчеты (переход в этот режим можно осуществить с помощью команды **Создание отчета в режиме конструктора**, отображаемой в окне БД).
- **Мастер отчетов** — автоматическое создание отчета на основе выбранных полей (переход в этот режим можно также осуществить с помощью команды **Создание отчета с помощью мастера**, отображаемой в окне БД).
- **Автоотчет: в столбец** — автоматическое создание отчета с полями, расположенными в один столбец.
- **Автоотчет: ленточный** — автоматическое создание ленточного отчета.
- **Мастер диаграмм** — создание отчета в виде диаграммы.
- **Почтовые наклейки** — создание отчета, отформатированного в виде для печати почтовых наклеек.

Наиболее широкие возможности по созданию отчета (по аналогии с формой) предлагает окно конструктора отчета (рис. 6.80).

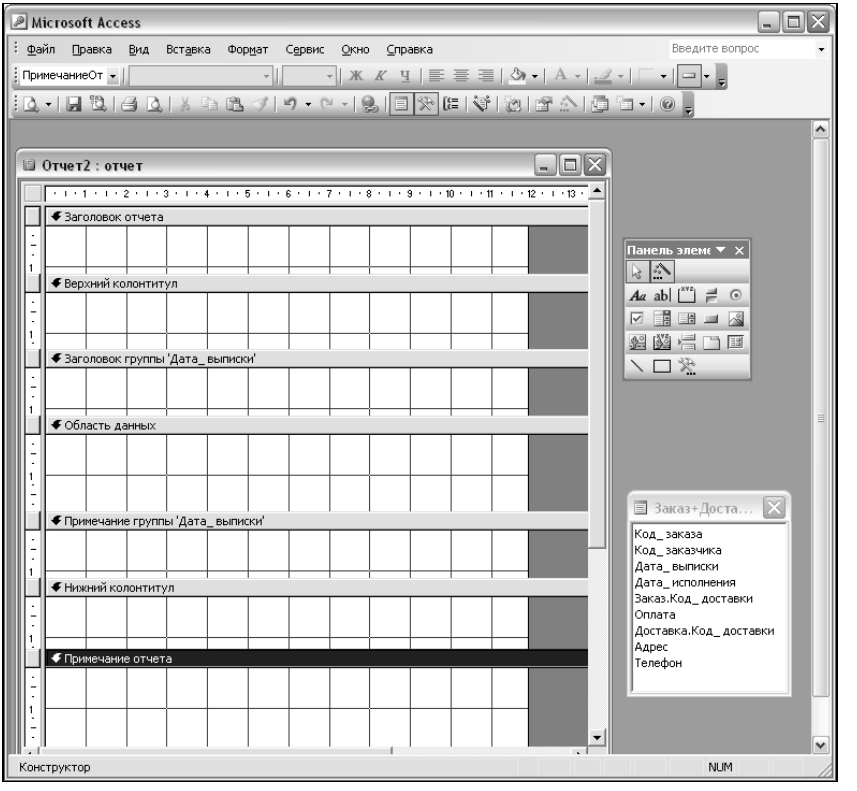


Рис. 6.80. Окно конструктора отчета

Создание и редактирование отчета происходят непосредственно в окне **Отчет** (рис. 6.80), в котором могут быть выделены следующие области:

- *заголовок отчета* (команда **Заголовок | Примечание отчета** в меню **Вид**) — располагается только один раз, перед верх-

ним колонтитулом, в отчете на титульной странице и содержит обычно заголовок, сведения о дате и времени;

- *верхний и нижний колонтитулы* (команда **Колонтитулы** в меню **Вид**) — располагаются по умолчанию на каждой странице отчета. В верхний колонтитул можно поместить информацию о полях, если отчет будет представлен в табличном виде. В верхний либо нижний колонтитулы можно поместить нумерацию страниц, логотип организации и т. п.;

- *заголовок группы* — располагается в начале новой группы записей, которая определяется общностью значений в группирующем поле. В заголовке группы может располагаться поле, которое содержит группу записей, а также заголовки полей данных, которые являются подуровнем для группы. Так как группа задается для определенного поля, то для отображения необходимой группы в окне конструктора отчетов необходимо выбрать команду **Сортировка и группировка** в меню **Вид** либо воспользоваться кнопкой  на панели инструментов **Конструктор отчетов**. В открывшемся диалоговом окне **Сортировка и группировка** (рис. 6.81) следует выбрать поля, по которым будет проводиться группировка данных, задать вид сортировки и определить области, выводимые в отчет (заголовок и примечание группы);

- *область данных* — область, предназначенная для размещения полей из таблицы или запроса, вычисляемых полей. Эта область выводит основную часть данных отчета;

- *примечание группы* — располагается в конце группы записей и используется, в основном, для помещения вычисляемых полей с использованием статистических функций (например, общее количество записей в группе);

- *примечание отчета* (команда **Заголовок | Примечание отчета** в меню **Вид**) — располагается только один раз в конце отчета и может содержать некоторую сводную информа-

цию обо всем отчете. При печати примечание отчета выводится перед нижним колонтитулом последней страницы.

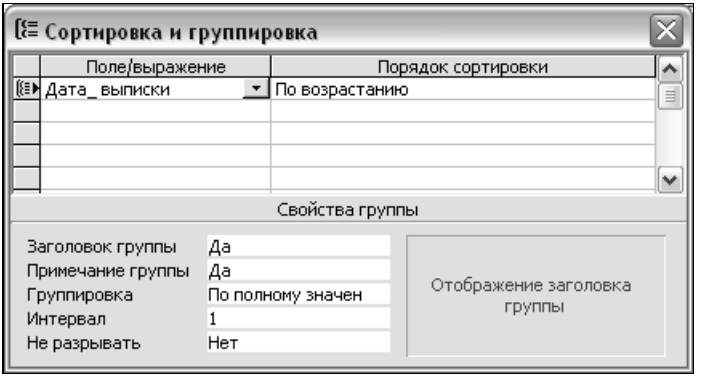


Рис. 6.81. Диалоговое окно **Сортировка и группировка**

Основные приемы по созданию отчета в режиме конструктора: размещение основных элементов в отчете (их редактирование, форматирование, изменение свойств и др.) аналогично приемам работы в конструкторе форм.

Следует обратить внимание на то, что использование вычисляемых полей в отчете может происходить в различных областях. В случае вычисления *для одной записи* необходимо вставить вычисляемое поле в область данных; *для группы записей* — в заголовок либо примечание группы; *для всех записей* — в заголовок либо примечание отчета.

Для формирования вычисляемого поля в окне **Свойства** этого поля на вкладке **Данные** вводятся необходимые выражения с использованием кнопки  .

Так как отчет предназначен, в первую очередь, для печатного представления, рекомендуется перед окончательным сохранением отчета просмотреть его общий вид и вид с разделением на страницы (соответственно команды **Образец** и **Предвари-**

тельный просмотр в меню **Вид**), а также выбрать параметры его размещения на странице (команда **Параметры страницы** в меню **Файл**). Печать отчета осуществляется с использованием команды **Печать** в меню **Файл**.

Пример. Разработать отчет для вывода всех заказов, сделанных конкретной фирмой. Итоговая информация по количеству сделанных заказов должна быть представлена в каждой группе "Заказчик" и в конце отчета.

Решение.

Результат решения примера показан на рис. 6.82.

Предварительно установлено, что для создания отчета `Заказчики и их заказы` необходимы следующие поля: `Название` (табл. 6.8), `Код_заказа`, `Дата_выписки` и `Дата_исполнения` (табл. 6.10), `Адрес_доставки` (табл. 6.9). В отчете необходимо выполнить группировку по полю `Название` (фирма заказчика), для каждой группы вычислить общее количество заказов, а также общее количество для всего отчета. Для оформления отчета можно использовать некоторую дополнительную информацию, которую можно поместить в колонтитулы.

Разработка отчета в соответствии с этим примером включает следующие этапы:

1. Так как в БД нет соответствующей таблицы или запроса, необходимо создать запрос на выборку данных — "Доставка заказов заказчикам", который содержит необходимые поля из табл. 6.8—6.10.
2. Перейти в режим конструктора отчета (выбрать, например, команду **Создание отчета в режиме конструктора**, отображаемой в окне БД).
3. Определить в свойствах отчета запрос "Доставка заказов заказчикам" в качестве источника записей на вкладке **Данные** (например, используя кнопку вызова системного меню формы в верхнем левом углу макета).

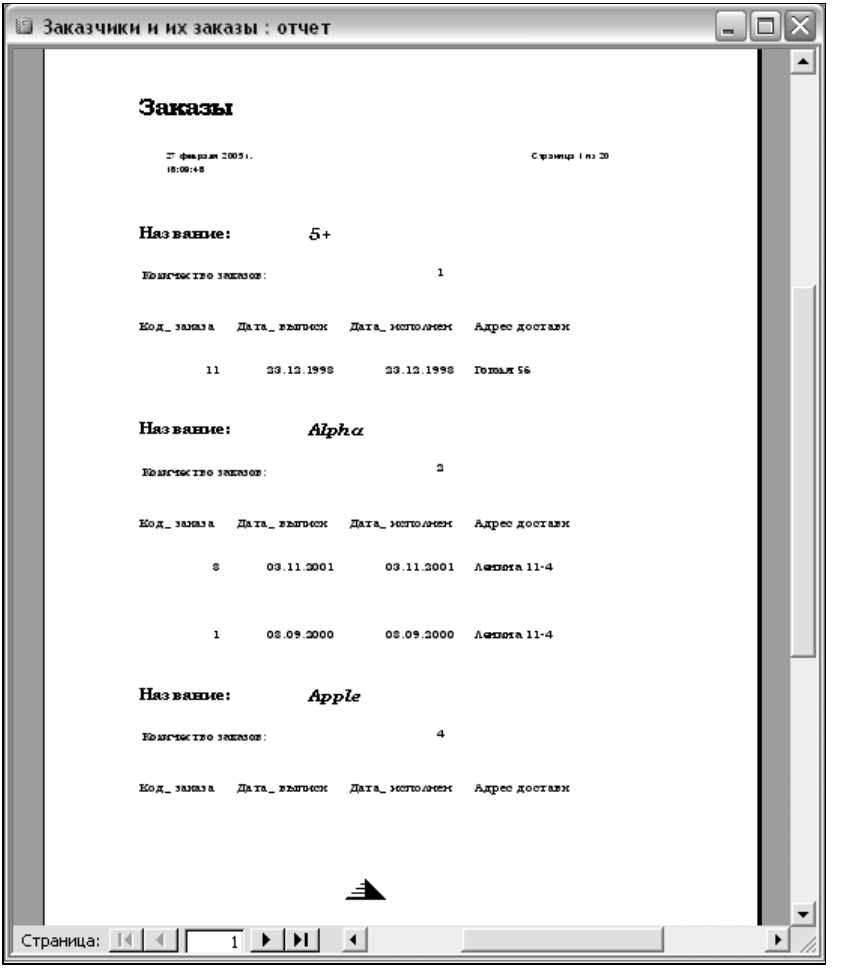


Рис. 6.82. Отчет "Заказчики и их заказы"

4. Вывести на экран: область заголовков и примечаний формы, линейку и сетку, панель элементов и список полей, а также область заголовка группы для поля `Название` (рис. 6.83), воспользовавшись соответствующими командами в меню **Вид**.

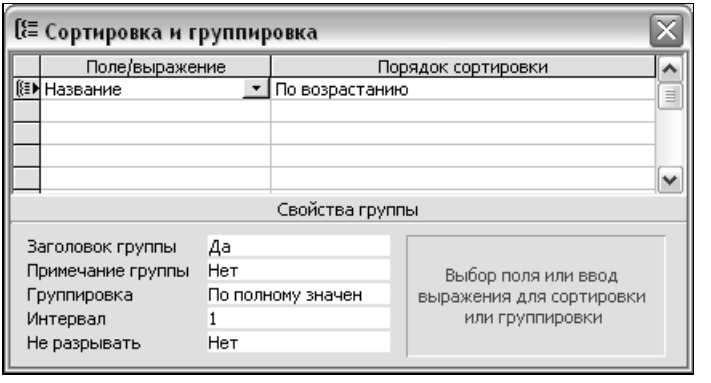


Рис. 6.83. Диалоговое окно **Сортировка и группировка**

5. Добавить в область данных с помощью мыши необходимые элементы:

   - *в заголовок отчета* — объект **Надпись** и ввести соответствующий текст (заказы), а также добавить дату и время;
   - *в верхний колонтитул* — поместить нумерацию страниц;
   - *в заголовок группы* — поместить поле `Название` (из списка полей), вычисляемое поле (подсчитывает количество заказов для каждой фирмы) и названия полей (`Код_заказа`, `Дата_выписки`, `Дата_исполнения`, `Адрес_доставки`);
   - *в область данных* — поместить поля `Код_заказа`, `Дата_выписки`, `Дата_исполнения`, `Адрес_доставки`;
   - *в нижний колонтитул* — поместить рисунок;
   - *в примечание отчета* — вычисляемое поле (подсчитывает количество заказов для всех фирм).

6. С помощью возможностей команд меню и панелей инструментов окна в режиме конструктора форм произвести необходимое редактирование и форматирование элементов, размещенных в области данных (рис. 6.84).

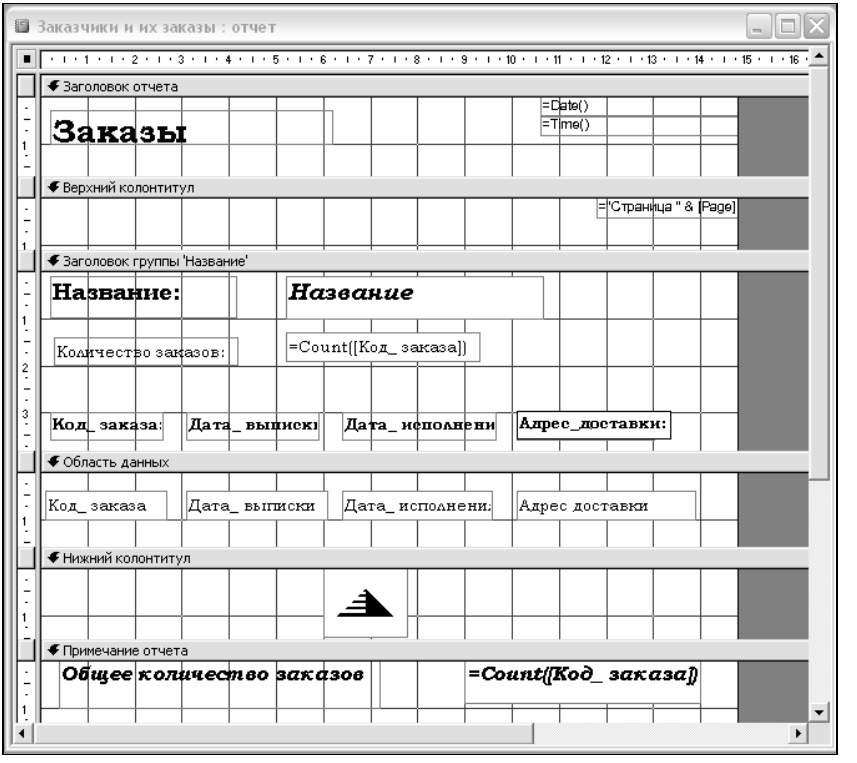


Рис. 6.84. Области разрабатываемого отчета с добавленными элементами

7. Просмотреть макет отчета, используя команды **Образец** и **Предварительный просмотр** в меню **Вид** либо кнопку (вид) на панели инструментов.
8. Сохранить отчет под именем "Заказчики и их заказы".

Некоторые сведения о макросах

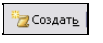
Макрос представляет собой автоматизированную последовательность команд, направленную на выполнение каких-либо действий.

Макросы позволяют решать следующие задачи:

- запускать вместе запросы, отчеты, формы, таблицы;
- открывать окна различных объектов БД;
- контролировать правильность ввода данных;
- перемещать данные;
- выполнять различные действия после щелчка на кнопке, к которой подключен макрос и др.

В MS Access используются макросы следующих типов:

- *линейные* — макросы, в которых команды расположены последовательно и выполняются по очереди;
- *макросы с условием* — макросы, в которых отдельные макрокоманды или их наборы выполняются в зависимости от некоторого условия;
- *групповые* — макросы, объединенные в группы с учетом некоторой логики использования.

Для создания и редактирования макросов в MS Access также предназначено специальное окно для конструирования макросов. Для открытия окна конструктора макросов следует воспользоваться кнопкой 

в окне БД при переходе к объектам макросы.

Окно конструктора макросов состоит из следующих частей (рис. 6.85):

- меню и панели инструментов, предназначенные для обработки макросов;
- панель макрокоманд (верхняя половина окна). По умолчанию на панель макрокоманд выводятся два столбца — **Макрокоманда** (для ввода макрокоманды, т. е. определенного действия с объектом) и **Примечание** (можно задать описание макроса, которое не влияет на его выполнение);
- панель аргументов (нижняя половина окна) — для задания аргументов (свойств) макрокоманд.

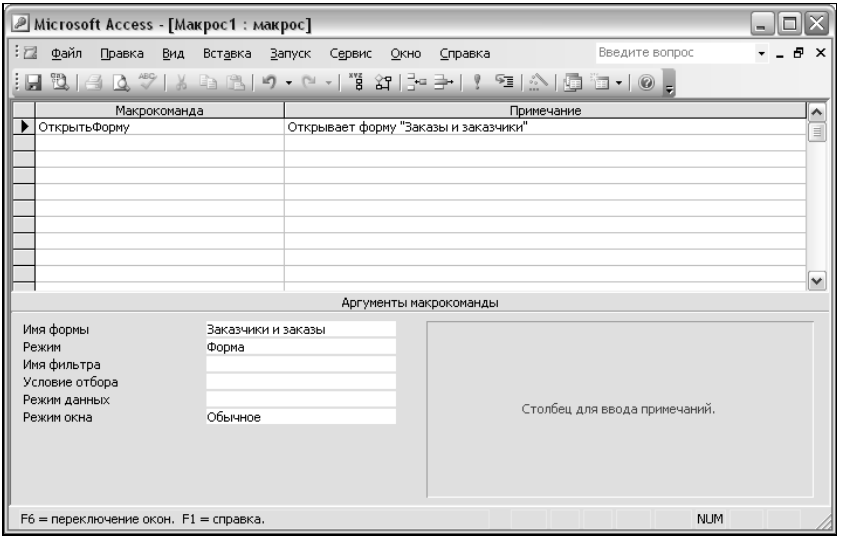


Рис. 6.85. Окно конструктора макросов

Если требуется создать макрос с условием, в панель макрокоманд добавляется столбец **Условие** (команда **Условия** в меню **Вид** либо кнопка  на панели инструментов **Конструктор макросов**). В случае работы с групповыми макросами следует добавить столбец **Имя макроса** (команда **Имена макросов** в меню **Вид** либо кнопка  на панели инструментов). В общем случае панель макрокоманд может содержать четыре столбца: **Имя макроса**, **Условие**, **Макрокоманда**, **Примечание** (рис. 6.86).

Макрокоманды вводятся в ячейки столбца **Макрокоманда** следующим образом:

- путем ввода их имени с клавиатуры;
- выбором из раскрывающегося списка (в ячейке столбца **Макрокоманда**);

- перетаскиванием требуемого объекта из окна БД в ячейку ввода макрокоманды (MS Access автоматически добавляет имя макрокоманды открытия соответствующего объекта БД).

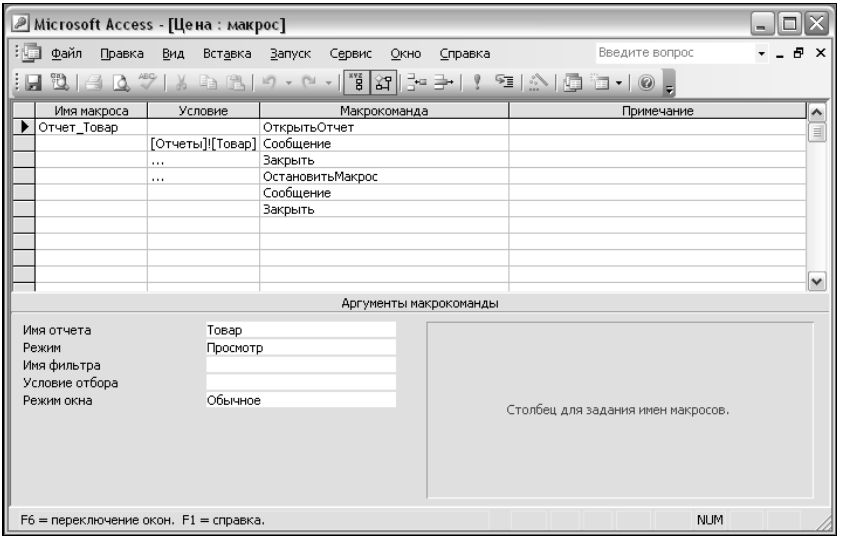


Рис. 6.86. Общий вид окна конструктора макросов

Основные приемы работы с макросами (ввод, редактирование, сохранение) аналогичны приемам работы с другими объектами MS Access.

После создания макроса его можно запустить на выполнение:

- в окне конструктора макросов (команда **Запуск** в меню **Запуск** либо кнопка );
- в окне БД в режиме макросы двойным щелчком на имени макроса;

- в окне любого активного окна (команда **Макросы** в меню **Сервис**);
- из другого макроса;
- с помощью кнопки панелей инструментов или команды меню, созданных для запуска этого макроса;
- с помощью комбинации клавиш, которые закреплены за макросом;
- с помощью различных элементов управления, помещенных в форму либо отчет и связанных с макросом (в свойствах объекта выполнение макроса связывается с определенным событием).

Пример. Создать макрос "Отчет_Товар", который связан с отчетом "Товар" и выводит сообщение о стоимости товаров.

Решение.

Результат решения примера представлен на рис. 6.86.

Для создания макроса "Отчет_Товар" необходимо:

1. Перейти в режим конструктора макросов (нажать кнопку 

в окне БД в режиме объекта макросы).
2. Добавить в панель макрокоманд столбец **Условие** и столбец **Имя макроса**.
3. Добавить в столбец **Имя макроса** `Отчет_Товар`.
4. Ввести в окно конструктора макросов информацию в соответствии с табл. 6.18.

Таблица 6.18. *Список макрокоманд для макроса "Отчет_Товар"*

Условие	Макрокоманда	Аргументы макрокоманды	
	Открыть отчет	Имя отчета	Товар
		Режим	Просмотр
		Режим окна	Обычное

Таблица 6.18 *(окончание)*

Условие	Макрокоманда	Аргументы макрокоманды	
`[Отчеты]![Товар]![Цена]>300000`	Сообщение	Сообщение	На складе есть товары, цена которых превышает 300 000 руб.
		Сигнал	Да
		Тип	Информационное
		Заголовок	Информация о цене
	Закрыть	Тип объекта	Отчет
		Имя объекта	Товар
		Сохранить	Подсказка
	Остановить макрос		
	Сообщение	Сообщение	Цена товаров на складе не превышает 300 000 руб.
		Сигнал	Да
		Тип	Информационное
		Заголовок	Информация о цене
	Закрыть	Тип объекта	Отчет
		Имя объекта	Товар
		Сохранить	Подсказка

5. Проверить выполнение макроса с помощью команды **Запуск** в меню **Запуск** либо кнопки  .
6. Сохранить макрос.

Придание приложению Microsoft Access законченного вида

Наряду с широкими возможностями по созданию, редактированию и использованию отдельных объектов, MS Access позволяет также связывать отдельные приложения в единое целое.

Создание и использование разнообразных форм (кнопочной, окон диалога и др.), панелей инструментов и меню, а также возможность управления автоматическим запуском MS Access позволяет разработать автоматизированное приложение, скрывающее отдельные объекты MS Access.

Для того чтобы разработать автоматизированное приложение, следует придерживаться следующих рекомендаций.

1. Разработать структуру и основные объекты БД.
2. Продумать последовательность действий в работе с окнами и определиться с основными формами, дающими доступ к нужным объектам БД (ввод и поиск информации, печать отчетов, модификации данных и др.).
3. Создать главную кнопочную форму (рис. 6.87), за каждой кнопкой которой закрепляется, как правило, открытие другого окна либо окон — уже с их помощью выполняется необходимая обработка данных. Следует заметить, что "нажатие кнопки" (свойство на вкладке **События**) связывается с выполнением макроса, который позволяет выполнить необходимые действия.
4. Создать необходимые панели инструментов и меню.
5. Задать автоматический запуск созданного приложения с использованием возможностей диалогового окна **Параметры запуска** (команда **Параметры запуска** в меню **Сервис**), показанного на рис. 6.88.

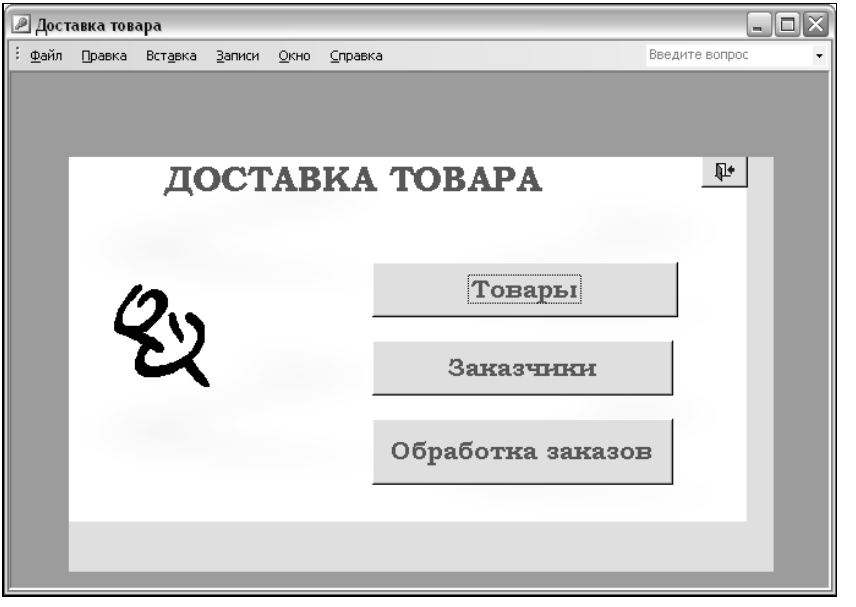


Рис. 6.87. Главная кнопочная форма БД "Доставка товара"

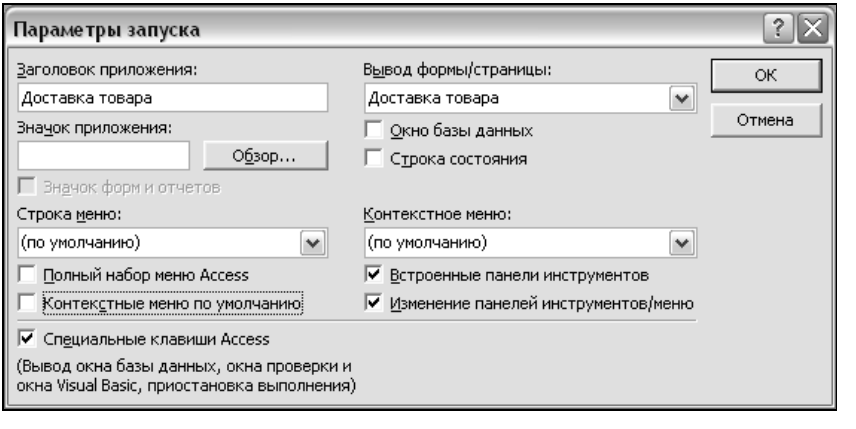


Рис. 6.88. Диалоговое окно **Параметры запуска**

Задания для самостоятельной работы

БД "Доставка товара"

Используя созданную при изучении этой главы БД "Доставка товара" выполнить следующие задания:

1. Сформировать запросы:
   - получить список заказов, отсортированный по дате выписки;
   - получить список товаров, цена которых находится в диапазоне от 10 000 руб. до 130 000 руб.;
   - определить количество заказов конкретного заказчика;
   - вывести построчное содержимое накладных;
   - вывести содержимое накладной по вводимому коду (параметрический запрос);
   - получить список заказчиков, у которых различаются физический адрес и адрес доставки;
   - получить список заказчиков, у которых в названии фирмы или организации есть цифра;
   - получить список заказчиков и их заказов, если заказчик приобретал товар определенного вида (например, бумага для ксерокса формата А4 или ксерокса формата А3);
   - получить стоимость товаров по заказам;
   - получить стоимость товаров по заказам, оформленным за конкретный промежуток времени;
   - получить суммарные стоимости товаров, приобретенных за конкретный промежуток времени, величина которых больше 25% от средней стоимости всех товаров;
   - стоимость товаров для конкретного заказчика.

2. Спроектировать формы:
   - для ввода, просмотра и поиска данных в базе;
   - для соответствующих разработанных запросов;
   - для учета, при необходимости, логически связанных объектов, а также возможности просмотра отчета.
3. Создать отчеты:
   - на основании информации, хранимой в базе, сформировать отчет в виде табл. 6.19.

Таблица 6.19. *Код накладной*

Название товара	**Цена, руб.**	**Количество**	**Стоимость**
Товар 1			
…			
Товар N			
Итого	XXXXX		

4. Создать макросы и главную кнопочную форму:
   - спроектировать главную кнопочную форму, используя рисунки и кнопки в качестве элементов управления. Кнопкам назначить макросы, открывающие созданные формы и отчет, а также поместить кнопку для выхода из приложения.

БД "Туристическое агентство"

Туристическое агентство предлагает разнообразные маршруты во многие страны мира. В каждой стране можно выбрать несколько маршрутов с различными целями путешествий (отдых, экскурсия, деловая поездка, лечение, шоп-тур, обуче-

ние и др.) и различными видами сервиса (наличие медицинской страховки, пансион, полупансион и т. д.). Для выявления наиболее популярных маршрутов и ведения определенной статистики необходимо создать БД.

1. Создать БД, используя табл. 6.20—6.24.

Таблица 6.20. *Страна*

Название поля	**Тип данных**	**Условия и ограничения**	**Примечание**
Код_страны	Числовой, целое	Уникальные значения (первичный ключ), не могут быть отрицательными	Используется для идентификации стран
Страна	Текстовый	Длина поля 100 символов	Название страны
Стоимость_визы	Денежный	Не могут быть отрицательными	Денежный эквивалент стоимости визы

Таблица 6.21. *Маршрут*

Название поля	**Тип данных**	**Условия и ограничения**	**Примечание**
Код_путевки	Числовой, целое	Уникальные значения (первичный ключ), не могут быть отрицательными	Используется для идентификации путевки
Название_маршрута	Текстовый	Длина поля 200 символов	Название маршрута
Цена_путевки	Денежный	Не могут быть отрицательными	Денежный эквивалент путевки

Таблица 6.22. *Сервис*

Название поля	**Тип данных**	**Условия и ограничения**	**Примечание**
Код_ сервиса	Числовой, целое	Уникальные значения (первичный ключ), не могут быть отрицательными	Используется для идентификации сервиса
Вид_ сервиса	Текстовый	Поле со списком	Название сервиса

Таблица 6.23. *Цель*

Название поля	**Тип данных**	**Условия и ограничения**	**Примечание**
Код_цели	Числовой, целое	Уникальные значения (первичный ключ), не могут быть отрицательными	Используется для идентификации цели поездки
Цель_ путешествия	Текстовый	Поле со списком	Название цели путешествия

Таблица 6.24. *Статистика продаж*

Название поля	**Тип данных**	**Условия и ограничения**	**Примечание**
№_пп	Счетчик	Уникальные значения (первичный ключ), не могут быть отрицательными	Используется для учета продаж
Дата_ продажи	Дата/ время	Краткий формат даты	Фиксируется дата продажи путевки
Код_путевки	Числовой, целое	Допускаются повторения при продажах различным лицам	Внешний ключ к табл. 6.21

Таблица 6.24 *(окончание)*

Название поля	Тип данных	Условия и ограничения	Примечание
Код_страны	Числовой, целое	Допускаются повторения при нахождении маршрута в одной и той же стране	Внешний ключ к табл. 6.20
Код_цели	Числовой, целое	Допускаются повторения при совпадении цели поездки	Внешний ключ к табл. 6.23
Код_ сервиса	Числовой, целое	Допускаются повторения при совпадении сервиса	Внешний ключ к табл. 6.22
Количество_ проданных_ путевок	Числовой, целое	Значения находятся в пределах от 1 до 50	Вводится количество проданных путевок

2. Связать таблицы в схему данных, используя связи "один-ко-многим", первичные и внешние ключи таблиц.
3. Сформировать следующие запросы:
   - запрос на вывод записей из базы, у которых цель путешествия "обучение" или "отдых";
   - параметрический запрос на вывод записей из базы данных "Путевки данной страны". Запрос происходит по параметру `Введите страну`. В выводимые поля обязательно включаются название маршрута, цена путевки, вид сервиса;
   - запрос на вывод записей из базы, у которых цена путевки находится в пределах от 900 у. е. до 2000 у. е.;
   - перекрестный запрос "Количество проданных путевок для стран за конкретный промежуток времени". В выводимые поля включить: название страны (заголовки строк), дату (заголовки столбцов, также записывается условие для вывода конкретного промежутка времени)

и количество проданных путевок (значение). Предполагается групповая операция для вывода общего количества проданных путевок за данный промежуток времени (заголовки строк);

- запрос с вычисляемым полем `Стоимость проданных путевок за конкретную дату`;
- запрос на создание таблицы `Путевки_для_лечения`. Включить эту таблицу в схему данных;
- запрос на обновление — уменьшить значение поля `Цена_путевки` на 30% для записей, у которых целью путешествия является лечение.

4. Спроектировать формы:
   - для ввода данных в базу;
   - для соответствующих разработанных запросов;
   - для учета, при необходимости, логически связанных объектов, а также возможности просмотра отчета.
5. Спроектировать отчет:
   - на основании информации, хранимой в базе, сформировать отчет в виде следующей таблицы:

Название страны	Стоимость визы, у. е.	Наименование маршрута	Цель путешествия	Цена путевки, у. е.	Количество проданных путевок	Стоимость проданных путевок, у. е.
Страна						
....						
Итого						
Итого по турагентству						

6. Создать макросы и главную кнопочную форму:

   - спроектировать главную кнопочную форму, используя рисунки и кнопки в качестве элементов управления. Кнопкам назначить макросы, открывающие созданные формы и отчет, а также поместить кнопку для выхода из приложения.

БД "Кинокомпания"

Некоторая кинокомпания хотела бы упорядочить и использовать с наилучшей выгодой информацию киноиндустрии. База данных должна включать информацию следующего плана: некоторые личные данные об актерах, которые хоть раз задействованы в каком-либо фильме; некоторые личные данные о режиссерах (включая тех, которые имеют хотя бы один фильм); обо всех фильмах, вышедших в прокат. При разработке базы данных учитывать также следующие замечания: задействованность актеров в соответствующем фильме; рейтинги актеров и режиссеров; цена проката фильма, прибыль от реализации; список всех жанров фильмов фиксирован. Кроме того, выводится следующая информация: занятость актера в фильмах; фильмы данного режиссера; рейтинг фильмов, актеров, режиссеров с учетом определенного года.

1. Создать базу данных, используя табл. 6.25—6.33.

Таблица 6.25. *Актеры*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ актера	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — да	Используется для идентификации актера

Таблица 6.25 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Фамилия	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Фамилия актера
Имя	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Имя актера
Год_ рождения	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Год рождения актера
Место_ рождения	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Место рождения актера
Национальность	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Национальность актера
Адрес	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Адрес актера

Таблица. 6.26. *Фильмы и актеры*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ фильма	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да	Внешний ключ к таблице Фильмы (табл. 6.27)

Таблица. 6.26 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
		(допускаются совпадения); вводятся из списка соответствующих значений табл. 6.27	
Код_актера	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.25	Внешний ключ к таблице `Актеры` (табл. 6.25)
Рейтинг_роли	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Рейтинг роли актера

Таблица. 6.27. *Фильмы*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_фильма	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — да	Используется для идентификации фильма
Название	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Название фильма
Жанр	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Жанр фильма

Таблица. 6.27 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Год_ выпуска	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Год выпуска фильма
Государство	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Государство, где произведен фильм
Комания_ производитель	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Компания — производитель фильма
Затраты_ на_производство	Денежный	Формат вывода: &#; условие на значение — не могут быть отрицательными; обязательное поле — да; индексированное поле — нет	Затраты на производство фильма
Краткая_ аннотация	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Аннотация к фильму

Таблица. 6.28. *Фильмы и режиссеры*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ фильма	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.27	Внешний ключ к таблице Фильмы (табл. 6.27)

Таблица. 6.28 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ режиссера	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.29	Внешний ключ к таблице `Режиссеры` (табл. 6.29)

Таблица. 6.29. *Режиссеры*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ режиссера	Числовой	Уникальные значения (первичный ключ); размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — да	Используется для идентификации режиссера
Фамилия	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Фамилия режиссера
Имя	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Имя режиссера
Год_ рождения	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Год рождения режиссера

Таблица. 6.29 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Место_ рождения	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Место рождения режиссера
Национальность	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Национальность режиссера
Адрес	Текстовый	Размер поля — 50 символов; обязательное поле — да; индексированное поле — нет	Адрес режиссера

Таблица. 6.30. *Рейтинг актеров*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ актера	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.25	Внешний ключ к таблице Актеры (табл. 6.25)
Год	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Год рейтинга
Рейтинг	Текстовый	Размер поля — 50 символов; мастер подстановок —1..10; обязательное поле — да; индексированное поле — нет	Рейтинг актера

Таблица. 6.31. *Рейтинг режиссеров*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ режиссера	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.29	Внешний ключ к таблице Режиссеры (табл. 6.29)
Год	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Год рейтинга
Рейтинг	Текстовый	Размер поля — 50 символов; мастер подстановок —1..10; обязательное поле — да; индексированное поле — нет	Рейтинг режиссера

Таблица. 6.32. *Рейтинг фильмов*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_ фильма	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.27	Внешний ключ к таблице Фильмы (табл. 6.27)

Таблица. 6.32 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Год	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Год рейтинга
Прибыль_от_реализации	Денежный	Формат вывода: $# ##0; условие на значение — не могут быть отрицательными; обязательное поле — да; индексированное поле — нет	Прибыль от реализации

Таблица. 6.33. *Разное*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Код_фильма	Числовой	Размер поля — длинное целое; обязательное поле — да; индексированное поле — да (допускаются совпадения); вводятся из списка соответствующих значений табл. 6.27	Внешний ключ к таблице Фильмы (табл. 6.27)
Дата	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	

Таблица. 6.33 *(окончание)*

Название поля	Тип данных	Условия и ограничения (свойства поля)	Описание
Цена_ проката_ видео	Денежный	Формат вывода: $# ##0; условие на значение — не могут быть отрицательными; обязательное поле — да; индексированное поле — нет	Цена проката видеокассет
Продано (кол-во)	Числовой	Размер поля — длинное целое; условие на значение — не могут быть отрицательными; обязательное поле — нет; индексированное поле — нет	Количество проданных видеокассет

2. Выполнить следующие запросы на выборку:

- получить все фильмы, снятые данным режиссером;
- отобразить все фильмы данного жанра и конкретной компании (параметрический запрос);
- вывести все убыточные фильмы за конкретный период;
- определить все фильмы, приносящие доход в определенном периоде;
- получить фильм конкретной компании-производителя со списком актеров и режиссеров;
- вывести список актеров (режиссеров), рейтинг которых совпадает с рейтингом на момент выхода фильма;
- отобразить список актеров, занятых в конкретном фильме, с учетом их ролевого рейтинга;
- получить список актеров и режиссеров заданной национальности (параметрический запрос);

- высчитать суммарную выручку по видеопрокату для конкретного фильма за конкретную дату `(столбец)(строка)`;
- вычислить доходы, принесенные конкретным фильмом `(фильм; год-столбец; прибыль-строка; затраты=доход<=значения)`.

3. Выполнить следующие запросы на изменение:
   - получить информацию о фильмах: фильм, дата, рейтинг (запрос на создание таблиц);
   - вывести информацию об актерах: ФИО, место рождения, национальность (запрос на создание таблиц);
   - увеличить затраты на производство на различные проценты в зависимости от жанра фильма (запрос на обновление);
   - уменьшить цену проката видео версии на 10% для фильмов, количество продаж которых больше 100.000 (запрос на обновление);
   - удалить из проката фильмы, приносящие убытки (запрос на удаление);
   - удалить из проката видеоверсии, продажи которых меньше 1000 (запрос на удаление);
   - добавить записи в таблицу, полученную в результате выполнения запроса на изменение № 1.
4. Создать несколько отчетов:
   - для вывода всех фильмов, произведенных конкретной компанией. Итоговая информация по количеству произведенных фильмов должна быть представлена в каждой группе "Компания-производитель" и в конце отчета;
   - для вывода имен и фамилий всех актеров, рожденных в определенный год и в заданном месте.

Информация должна быть представлена в виде следующей таблицы:

Имя актера	Фамилия	Год рождения	Место рождения

- на основании информации, хранимой в базе, сформировать отчет в виде:

Название фильма	Цена проката, $	Количество	Стоимость
Фильм 1			
…			
Фильм N			
Итого	XXXXX		

Глава 7

Создание клиент-серверных приложений средствами InterBase и Delphi

В настоящее время под клиент-серверной архитектурой понимается модель, в которой доступ к удаленной базе данных осуществляется с помощью компьютера. В системе "клиент-сервер" обязательно наличие сети (локальной либо глобальной). В ней один из компьютеров является сервером, располагающим информационно-вычислительными ресурсами (например, процессоры, файловая система, почтовые ресуры, ресурсы печати, база данных), остальные — клиентами, которые используют данные ресурсы. Для архитектуры "клиент-сервер" характерны отличительные особенности, касающиеся как проетирования системы, так и ее реализации. Данная глава посвящена рассмотрению некоторых аспектов, касающихся создания клиент-серверных приложений.

Принципы создания клиент-серверных приложений

Для взаимодействия клиентской программы с данными нужно использовать утилиту, которая обеспечивает клиентскую программу необходимыми данными. Если для создания кли-

ентского приложения выбрана среда Delphi, то в качестве такой утилиты можно использовать BDE (Borland Database Engine) Administrator (либо SQL Explorer).

Замечание

В Delphi 7.0 для связи клиенсткой программы с данными используется компонент IBDatabase.

Двухзвенная архитектура "клиент-сервер"

Утилита BDE Administrator не является частью приложения. В зависимости от СУБД она может располагаться на машине клиента либо сервера.

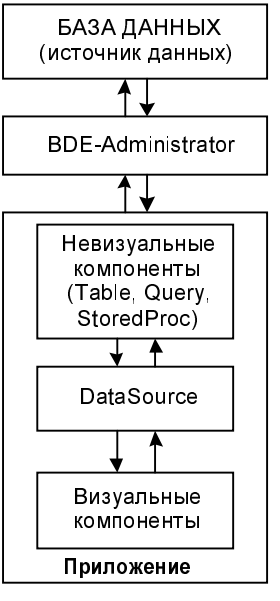


Рис. 7.1. Схема взаимодействия программ с данными

Как правило, клиентское приложение использует BDE Administrator для непосредственного обращения к данным. Кроме того, между приложением и данной утилитой имеется ряд компонентов, которые упрощают разработку программы (рис. 7.1).

Невизуальные компоненты осуществляют непосредственную работу с BDE, три из которых (`Table`, `Query`, `StoredProc`) служат источниками данных. Визуальные компоненты отображают поставляемые ими данные и помогают создавать удобный интерфейс пользователя. Между источниками данных и визуальными компонентами обязательно располагаются промежуточные компоненты `DataSourse`, открывающие либо закрывающие потоки данных, которыми обмениваются источники данных с визуальными компонентами.

В архитектуре "клиент-сервер" (рис. 7.2) между BDE Administrator и базой данных располагается *сервер баз данных* (специальная программа, управляющая базой данных).

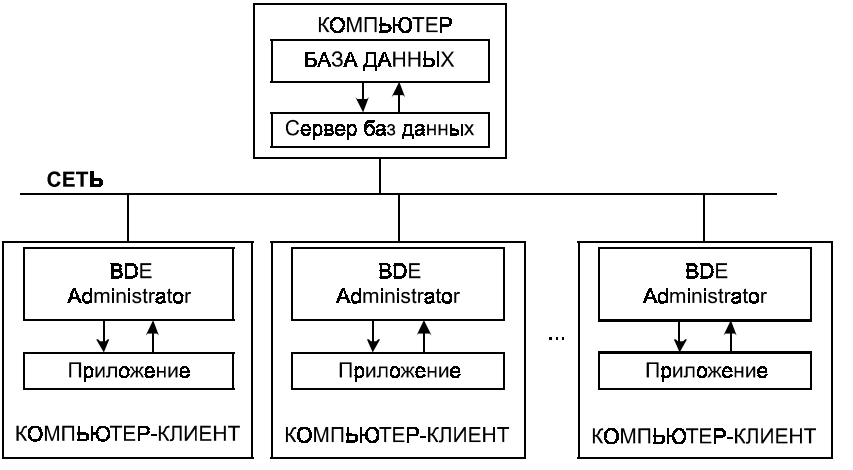


Рис. 7.2. Двухзвенная архитектура "клиент-сервер"

В архитектуре "клиент-сервер" используются различные промышленные серверы баз данных, например InterBase, Oracle, MS SQL Server, Sybase и т. д.

Таким образом, двухзвенная архитектура "клиент-сервер" включает: программу клиента (первое звено), сервер баз данных и базу данных (второе звено).

При создании клиентских приложений, которые используют двухзвенную архитектуру, необходимо придерживаться следующих рекомендаций.

- На стороне сервера:
  - установить сервер баз данных;
  - создать файл базы данных, который размещается на сервере (определить место размещения базы данных на диске и задать название базы данных);
  - создать необходимые объекты базы данных.
- На стороне клиента:
  - с помощью утилиты BDE Administrator (либо другой) задать псевдоним и установить соответствующие настройки для обращения к удаленной базе данных;
  - реализовать клиентское приложение, например, средствами Delphi, используя соответствующие компоненты (см. рис. 7.1 и табл. 7.1):
    - ◊ `Database`, `Session` — для связи с удаленной базой данных;
    - ◊ невизуальные компоненты (`Table`, `Query`, `StoredProc`);
    - ◊ компоненты `DataSource` (по одному на каждый источник данных);
    - ◊ визуальные компоненты (например, `TBGrid` — сетка (таблица данных), `TBNavigator` — компонент-навигатор и т. д.).

Трехзвенная архитектура "клиент-сервер"

В трехзвенной архитектуре "клиент-сервер" (N-tier или multi-tier архитектура) создается дополнительное вспомогательное приложение, в которое включаются все компоненты — источники данных, которые в двухзвенной архитектуре располагаются на стороне клиентского приложения, а также компоненты `TDatabase` и `TSession`. Затем данное приложение регистрируется в качестве COM- или CORBA-сервера на всех компьютерах клиента, после чего оно становится *сервером приложений*. В данном случае клиентские машины могут не иметь BDE, а клиентские программы уже не включают коды компонентов-источников и других вспомогательных компонентов. Для получения доступа к серверным данным клиентские машины обращаются к удаленному (находящемуся на другой машине) серверу приложений, который реализует необходимый обмен данными (рис. 7.3).

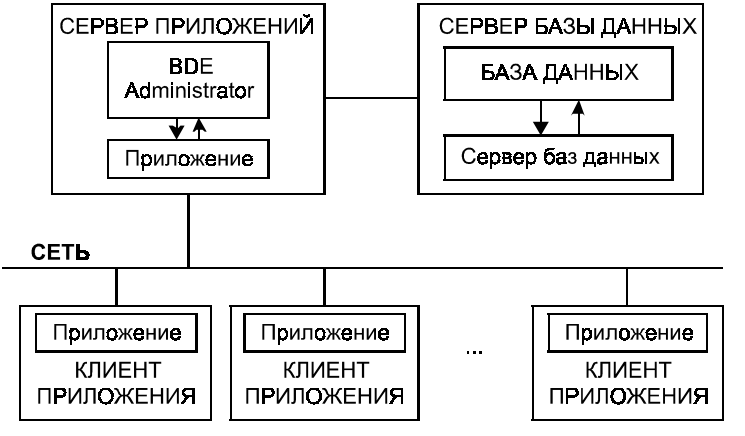


Рис. 7.3. Трехзвенная архитектура "клиент-сервер"

Замечание

COM (Component Object Model) — компонентная модель объекта корпорации Microsoft. Технология COM предназначена для связи клиентского приложения с приложением сервера. Основной чертой COM-объекта является интерфейс, который имеет уникальный номер и набор параметров, описывающих методы, события и свойства общего объекта. Получив интерфейс внешнего COM-объекта, приложение клиента может его использовать как собственный.

CORBA (Common Object Request Broker Architecture) — архитектура с поставщиком требуемых общих объектов независимой группы OMG. Технология CORBA также использует интерфейс объекта, но с помощью интегрированного слоя, осуществляющего доступ к удаленным объектам.

Следует заметить, что сервер приложений может располагаться на любой сетевой машине, на которой есть BDE. В этом случае каталог его размещения должен быть доступен другим сетевым машинам, а сама машина сервера приложений должна быть включена в период работы с сервером данных.

С помощью словарей базы данных можно перенести в компоненты-источники и связанные с ними поля часть бизнес-правил, касающуюся различных ограничений на значения вводимых данных. В этом случае неправильные данные будут отвергаться сервером приложений и не будут передаваться в сервер БД.

В клиентской программе, которая в трехзвенной архитектуре называется *тонким клиентом*, размещается клиентский набор данных, представляющий собой копию части данных из БД. Все изменения, которые пользователь вносит в данные, изменяют эту локальную копию и могут до нужного времени передаваться в БД (режим отложенной обработки данных). Кроме того, при работе с большими таблицами можно потребовать от сервера приложений передавать в локальный набор записи таблицы порциями, достаточными для одновременного отображения на экране клиента, что снижает загрузку сети и, следовательно, уменьшает время ожидания результата запроса.

При создании клиентских приложений, которые используют трехзвенную архитектуру, необходимо придерживаться следующих рекомендаций.

- На стороне сервера:
  - установить сервер баз данных;
  - создать файл базы данных, который размещается на сервере (определить место размещения базы данных на диске и задать название базы данных);
  - создать необходимые объекты базы данных.
- На стороне сервера приложений:
  - с помощью утилиты BDE Administrator (либо другой) задать псевдоним и установить соответствующие настройки для обращения к удаленной базе данных;
  - реализовать приложение, например, средствами Delphi с использованием соответствующих компонентов: (`Database`, `Session`) — для связи с удаленной базой данных; (`Table`, `Query`, `StoredProc`) — компоненты — источники данных, `Provider` — компонент, обеспечивающий связь каждого данного источника с клиентом (по одному на каждый источник данных).
- На стороне клиента:
  - реализовать клиентское приложение, например, средствами Delphi, используя соответствующие компоненты:
    - ◊ компонент-коннектор (в зависимости от используемого протокола), например, `TDCOMConnection`;
    - ◊ `TClientDataSet` — компонент-посредник между компонентом-источником сервера приложений (по одному на каждый источник данных);
    - ◊ компоненты `DataSource` (по одному на каждый источник данных);
    - ◊ визуальные компоненты.

Таблица 7.1. *Некоторые компоненты Delphi 7.0 для работы с базами данных*

Компонент	Назначение
`Database`	Активно используется при работе в архитектуре "клиент-сервер". Позволяет соединяться с удаленной БД и управлять параметрами соединения, получать информацию о БД, получать информацию об открытых наборах данных и о доступных таблицах БД
`DataSource`	Служит промежуточным звеном для соединения визуальных компонентов с компонентами-источниками. Позволяет устанавливать некоторые параметры набора данных, устанавливать состояние набора данных и отслеживать в нем изменения
`Query`	Реализует набор данных, источником для которого является одна или несколько таблиц БД. Структура записи набора данных, состав набора данных определяются SQL-запросом. Используются для групповых операций обновления, добавления или удаления в таблицах БД, а также выполняет любые другие действия, предусмотренные реализацией языка SQL для той СУБД, с которой работает `TQuery`
`Session`	Содержит информацию о текущем сеансе работы с БД. Позволяет определить список доступных БД, открывать, отыскивать и закрывать БД, управлять параметрами сеанса
`StoredProc`	Используется в архитектуре "клиент-сервер" для доступа к хранимым процедурам, расположенным на сервере БД. Хранимые процедуры кодируются с помощью особого процедурного языка, хранят, как правило, часто употребляемые запросы к БД и могут разделяться между различными приложениями. Компонент `TStoredProc` наряду с компонентами `TTable` и `TQuery` является набором данных, поскольку может возвращать множество записей из одной или нескольких физических таблиц БД
`Table`	Реализует набор данных, источником для которого является одна таблица БД. Содержит множество методов, свойств и событий, посредством которых программа оперирует с данными

Основные возможности сервера баз данных InterBase

Промышленный сервер баз данных InterBase предназначен для хранения и обработки большого объема информации в условиях одновременной работы с базой данных многих клиентских приложений.

Средства InterBase позволяют:

- определить схему базы данных; для задания ссылочной целостности используются первичный и внешние ключи, ограничения на значения отдельных столбцов (вводимый диапазон, соответствие маски ввода и т. д.), триггеры (подпрограммы, автоматически выполняемые сервером до и/или после события изменения записи в таблице базы данных), генераторы (для создания и использования уникальных значений нужных полей);
- определять хранимые процедуры для ускорения работы клиентских приложений с удаленной базой данных;
- определять пользовательские функции, в которых могут реализовываться различные возможности обработки данных, отсутствующие в стандартных встроенных функциях;
- посылать уведомления клиентским приложениям о наступлении какого-либо события;
- создавать представления.

Утилита IBConsole (InterBase Console)

Утилита InterBase Console (рис. 7.4) служит для:

- управления сервером;
- регистрации пользователей и установки для них прав доступа к тем или иными БД;

- проверки целостности БД;
- сохранения и восстановления БД;
- сборки мусора;
- восстановления потерянных транзакций для БД, расположенных на разных серверах;
- получения статистической информации.

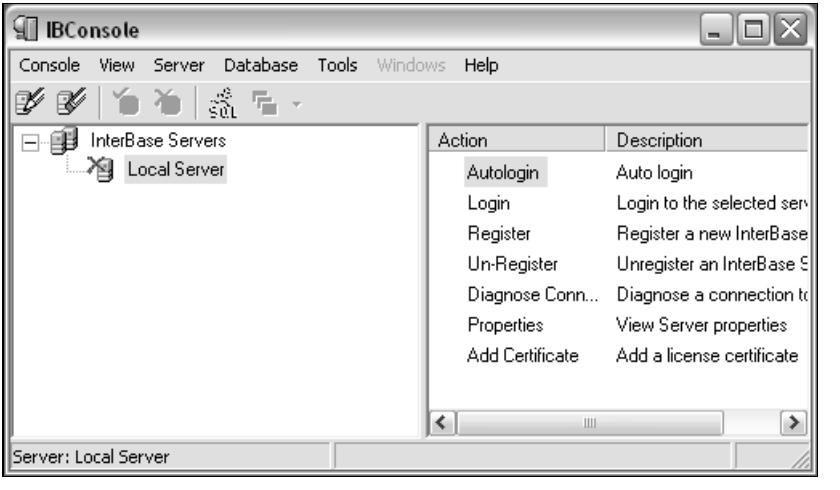


Рис. 7.4. Утилита IBConsole

Соединение с сервером

С помощью IBConsole можно работать как с локальными, так и с удаленными данными. Следует помнить, что на одной рабочей машине может быть размещен только один локальный сервер и зарегистрировано несколько удаленных серверов. В случае регистрации очередного сервера утилитой IBConsole автоматически предлагается зарегистрировать удаленный сервер (Remote Server). Для соединения с сервером следует выбрать команду **Server | Register** (рис. 7.5).

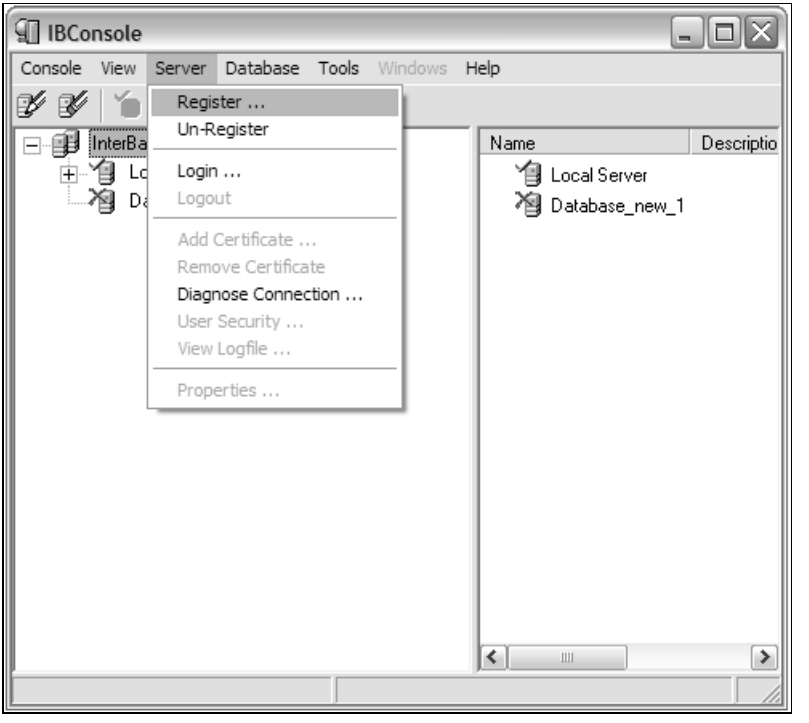


Рис. 7.5. Выбор команды для регистрации локального сервера либо для соединения с удаленным сервером базы данных

В окне **Register Server and Connect** (рис. 7.6) следует указать имя сервера **(Server Name)**, протокол **(Network Protocol)**, псевдоним, используемый для сервера на данной машине **(Alias Name** — прописывается, например, в утилите BDE Administrator), описание **(Descriptions)** — если необходимо, а также имя пользователя **(User Name)** и пароль **(Password**; имя пользователя — SYSDBA, пароль — masterkey в случае начала работы с утилитой; в дальнейшем можно изменить имя пользователя и пароль). Если выбирается режим локального доступа к данным, активными в данном окне будут имя пользователя и пароль.

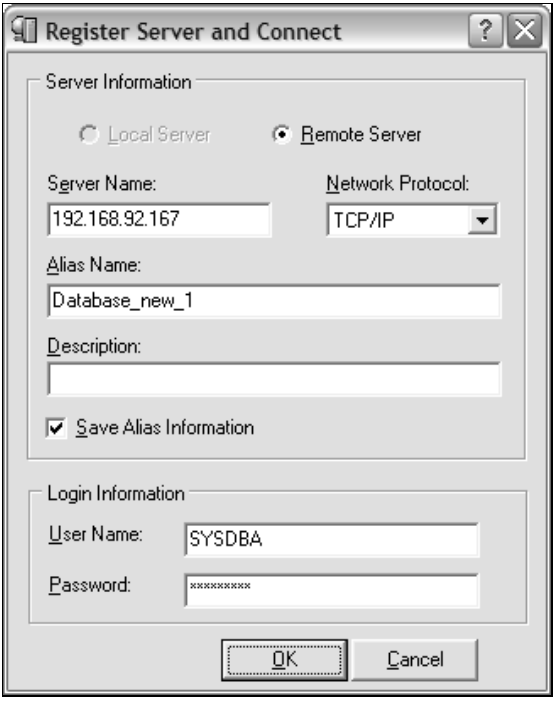


Рис. 7.6. Регистрация удаленного сервера в окне **Register Server and Connect**

Создание базы данных

Для создания базы данных следует выполнить команду **Database | Create Database** (рис. 7.7). В открывшемся окне **Create Database** (рис. 7.8) нужно задать псевдоним для базы данных **(Alias)**, прописать путь нахождения файла базы **(Filename(s))** и его первоначальный размер **(Size(Pages))**, а также указать некоторые другие опции **(Options**: размер для страницы хранения данных, кодировку символов, диалект SQL).

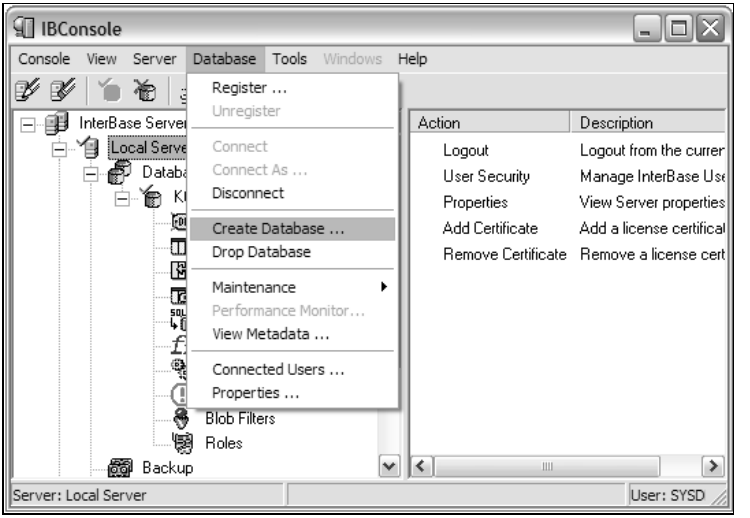


Рис. 7.7. Выбор команды для создания базы данных

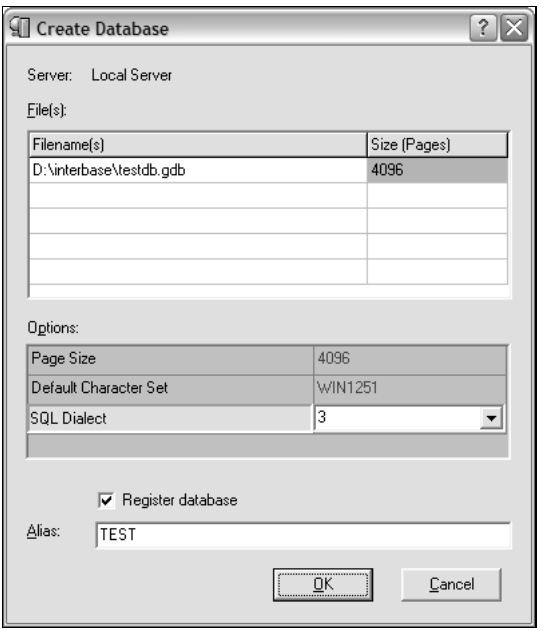


Рис. 7.8. Окно **Create Database**

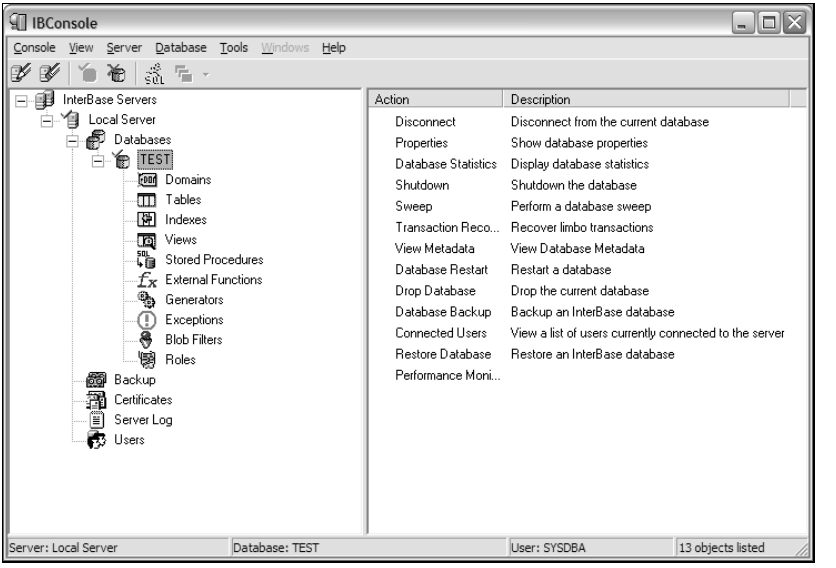


Рис. 7.9. Объекты базы данных в окне **IBConsole**

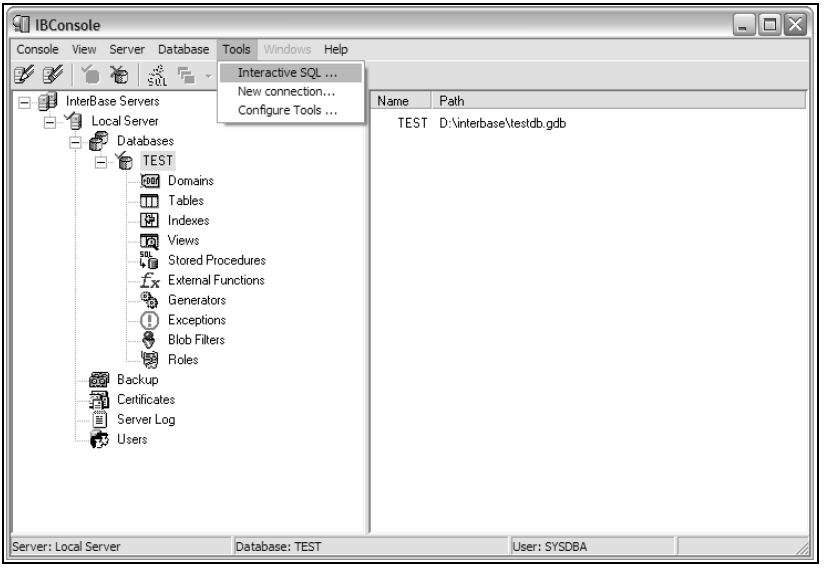


Рис. 7.10. Выбор команды для вызова утилиты Interactive SQL

После создания базы данных имеется возможность просмотреть все ее объекты (рис. 7.9).

Создание таблиц, представлений, доменов и других объектов БД, а также выполнение запросов к БД можно осуществлять утилитой Interactive SQL (команда **Tools | Interactive SQL**, рис. 7.10 и 7.11).

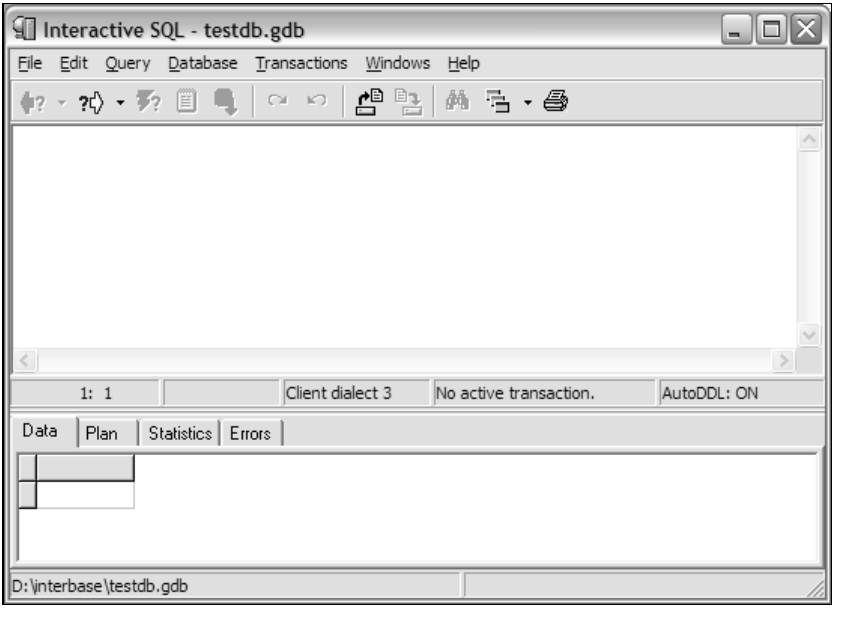


Рис. 7.11. Окно утилиты Interactive SQL

Соединение с базой данных

Для соединения с БД следует выбрать команду меню **Database | Register** (см. рис. 7.7) и указать имя БД и необходимые опции (рис. 7.12).

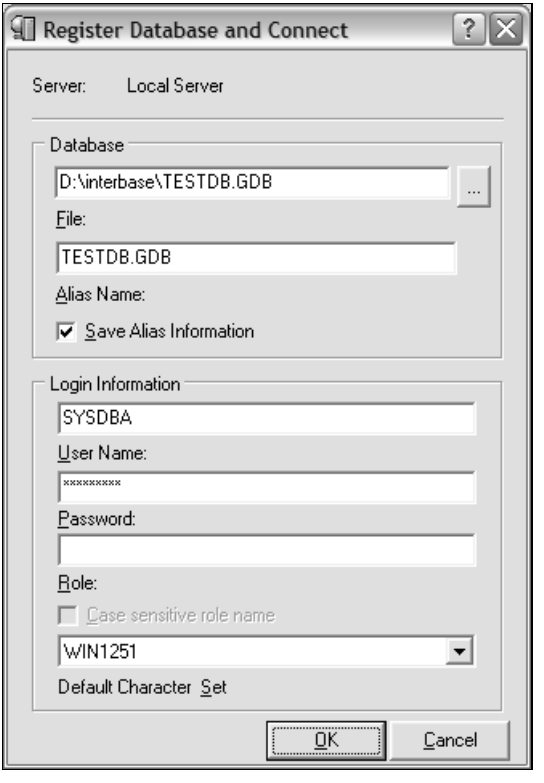


Рис. 7.12. Окно **Register Database and Connect**

Выбор текущего сервера и базы данных

В окне **IBConsole** одновременно можно установить соединение с несколькими серверами и базами данных. Для выбора конкретного сервера или БД следует выделить необходимый объект в левой части окна **IBConsole** и произвести соединение сначала с сервером, а затем с необходимой БД (рис. 7.13).

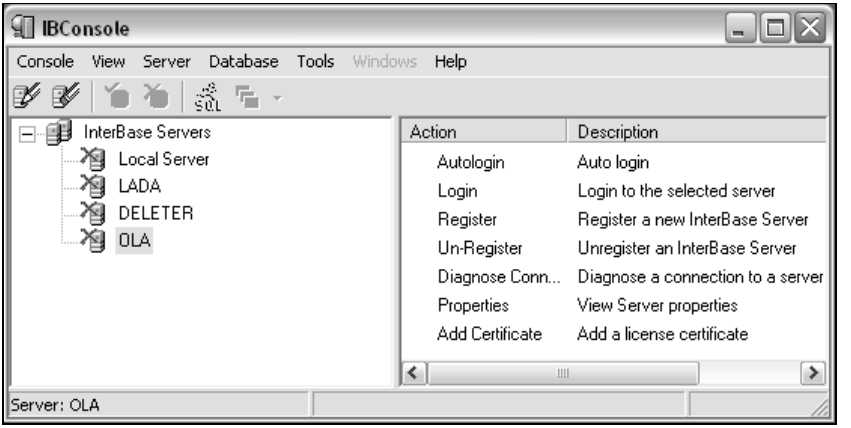


Рис. 7.13. Просмотр серверов (локального и удаленных) баз данных

Разрыв соединения

Для разрыва соединения с БД следует воспользоваться командой **Database | Disconnect**, а для разрыва соединения с сервером — командой **Server | Logout**. В последнем случае будут разорваны все соединения с БД этого сервера.

Изменение свойств базы данных

Свойства текущей, т. е. рабочей, БД можно изменить, выполнив команду меню **Database | Properties** (рис. 7.14). На вкладке **Alias** производится смена псевдонима для текущей БД, а на вкладке **General** — установка других необходимых опций (принудительная запись, интервал очистки, диалект SQL для данной БД, только чтение).

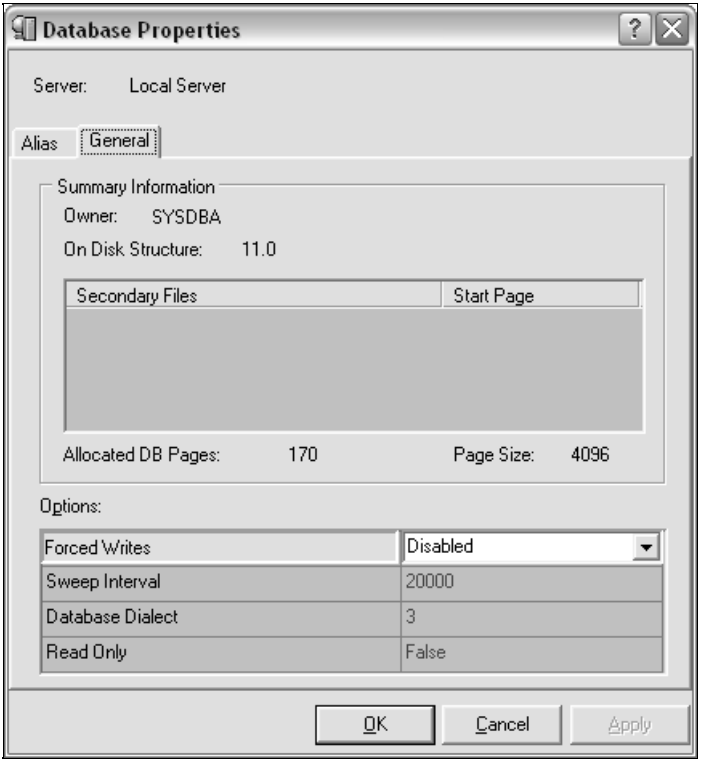


Рис. 7.14. Окно **Database Properties**

Статистические данные о базе данных

Для запуска процесса сбора статистики следует воспользоваться командой **Database | Maintenance | Database Statistics**. В появившемся окне необходимо указать опцию для вывода статистических данных.

После выбора соответствующей опции в окне **Database Statistics** (рис. 7.15) можно получить статистический отчет по конкретной базе данных (рис. 7.16).

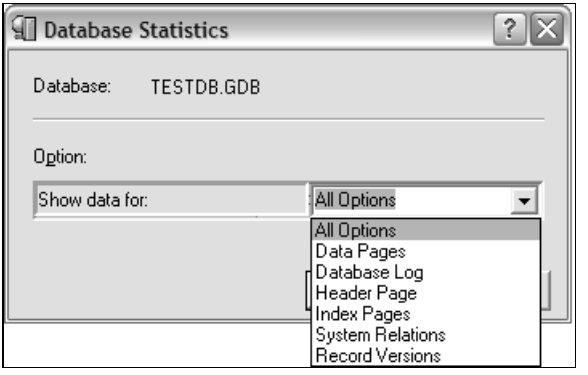


Рис. 7.15. Окно **Database Statistics**

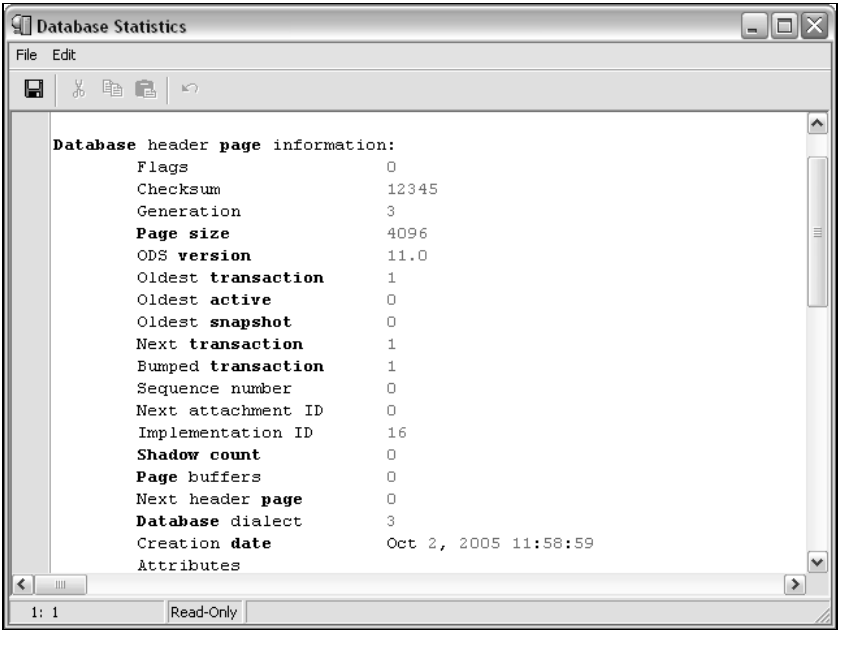


Рис. 7.16. Вывод статистических данных о базе данных в окне **Database Statistics**

Основные данные о статистическом отчете базы данных представлены в табл. 7.2.

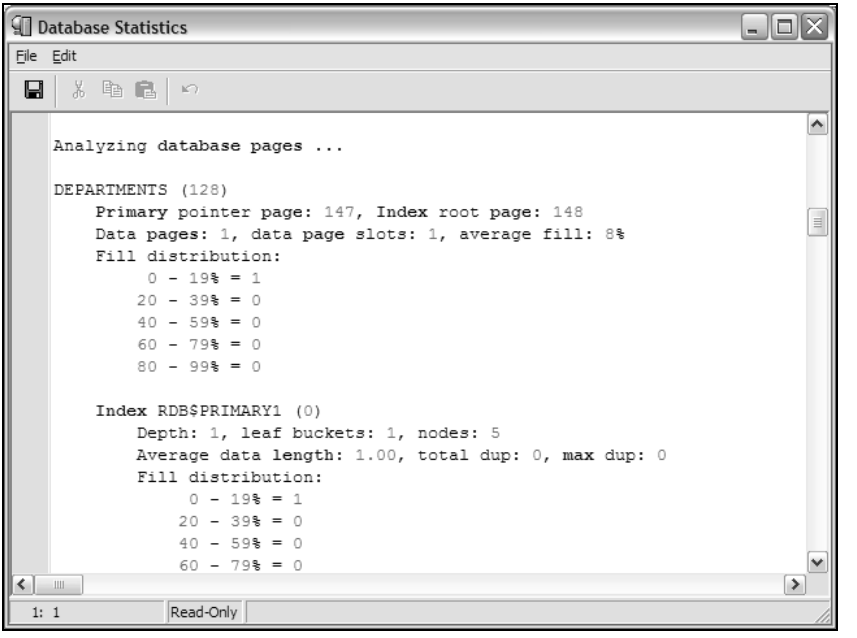


Рис. 7.17. Информация о таблицах базы данных в окне **Database Statistics**

Таблица 7.2. *Разделы статистического отчета по базе данных*

Опция	Описание
Раздел **Database header page information** содержит сведения из заголовочной страницы БД	
Flags	Указывает флаг БД. Некоторые значения: 1 — БД является "зеркальной" копией основной БД; 2 — разрешен режим принудительной записи (Forced Writes), когда запись данных производится в физической БД; при отмене этого режима запись выполняется

Таблица 7.2 *(продолжение)*

Опция	Описание
	в буфер, а потом в фоновом режиме (обычно при переполнении буфера) переносится на диск; при сбое системы данные из буфера могут быть потеряны, что может привести к непредсказуемым последствиям
Checksum	Контрольная сумма заголовка БД. Уникальное значение, которое вычисляется по всем данным в заголовке БД. Используется для анализа правильности данных в заголовке БД
Generation	Счетчик, который увеличивается на 1 при каждом обновлении данных в заголовке БД
Page size	Размер страницы БД в байтах
ODS version	Версия структуры БД на диске
Oldest transaction	Номер старейшей незавершенной транзакции (см. также **Next transaction**). Незавершенной считается транзакция, если она активна, отменена (rolled back) или зависла (in limbo, т. е. во время ее действия и до применения к ней подтверждения или отката произошел сбой, и после него невозможно сказать, завершена транзакция или нет; такое возможно для транзакций, охватывающих БД, которые расположены на различных серверах)
Oldest active	Самая старая активная транзакция
Next transaction	Номер, который будет присвоен следующей транзакции. При выполнении условия: `Next transaction - Oldest transaction > Sweep interval` производится автоматическая чистка мусора в БД. `Sweep interval` (по умолчанию 20 000) — число транзакций, через которое происходит автоматическая чистка мусора

Таблица 7.2 *(продолжение)*

Опция	Описание
Sequence number	Номер первой страницы БД
Next attachment ID	Номер следующего соединения с БД
Number of cache buffers	Размер буфера в страницах БД
Next header page	Номер следующей страницы заголовка БД
Creation date	Дата создания БД
Attributes	Атрибуты БД
Раздел **Database file sequence** описывает характеристики последовательности файлов, из которых состоит БД	
Раздел **Database log page information** содержит информацию о страницах журнала БД (только для серверов NetWare)	
Раздел **Analyzing database pages** включает информацию обо всех таблицах и индексах БД (см. рис. 7.17). Для каждой таблицы базы данных выводится следующая информация:	
Primary pointer page	Номер начальной страницы таблицы БД
Index root page	Номер начальной страницы для хранения индексов таблицы БД
Data pages	Общее число страниц для хранения данных
Average fill	Процент заполнения страниц для хранения данных
Fill distribution	Диаграмма заполнения страниц для хранения данных
Index	Имя индекса
Depth	Число уровней в дереве индексных страниц (оптимальный показатель — не больше 3. При глубине индекса (depth) больше 3 сортировка с его использованием становится неэффективной)

Таблица 7.2 *(окончание)*

Опция	Описание
Leaf buckets	Число страниц нижнего уровня в дереве индексных страниц
Nodes	Общее число страниц в дереве
Average data length	Средняя длина каждого ключа в байтах
Total dup	Общее число строк индекса с дублированными значениями индексных полей
Max dup	Число строк индекса с максимальным числом дублированных значений индексных полей
Fill distribution	Гистограмма, показывающая число индексных страниц, соответствующих определенному проценту заполнения

Сборка мусора

Необходимость сборки мусора обусловливается следующими причинами. При чтении транзакцией записи все старые версии этой записи, не использующиеся активными транзакциями, удаляются. Это касается только записей, созданных подтвержденными транзакциями. Версии, созданные транзакциями, впоследствии откатившимися (rolled back), а также старые версии записей, закрепленные за активными транзакциями, остаются в БД. Это сильно увеличивает объем БД и понижает быстродействие работы с ней. Поэтому в целях оптимизации используют *сборку мусора* (garbage collection).

Для принудительной сборки мусора следует выбрать команду **Database | Maintenance | Sweep**.

Автоматическое изменение значения **sweep interval**, использующегося для включения автоматической сборки мусора, может быть произведено командой **Database | Properties |**

General. Однако следует помнить о том, что автоматическая сборка мусора может быть неэффективной, если в момент ее осуществления имеется много активных транзакций вследствие интенсивной работы клиентов с БД. В этом случае много версий записей, занятых активными транзакциями, не войдет в число "собранных" записей; кроме того, сборка мусора существенно замедлит выполнение активных транзакций. В силу этого рекомендуется собирать мусор либо вручную при минимальном числе пользователей, либо при восстановлении БД (когда подключений к БД быть вообще не должно). Просмотреть активных пользователей можно, выбрав команду **Database | Connected Users**.

Создание резервной копии (сохранение) и восстановление базы данных

Сохранение БД необходимо для:

- получения текущей резервной копии БД, которая может быть использована для восстановления БД в случае ее сбоя;
- улучшения характеристик производительности при работе с БД.

Переход в однопользовательский режим соединения с базой данных

Создание резервной копии может выполняться без отключения других пользователей. Однако следует помнить, что в данном случае в резервной копии будут сохранены все данные, измененные на момент создания резервной копии. Поэтому резервное копирование обычно осуществляют в момент отсутствия соединений с БД со стороны пользователей. Как правило, это происходит в вечерние, ночные либо утренние часы. Существующие соединения пользователей могут быть принудительно разорваны. Для этого следует выбрать

команду **Database | Maintenance | Shutdown** и в диалоговом окне **Database Shutdown** установить параметры принудительного разъединения (рис. 7.18).

Рис. 7.18. Окно **Database Shutdown**

В окне **Database Shutdown** можно выбрать следующие параметры перехода в однопользовательский режим:

- **Deny new connections while waiting** — все существующие соединения пользователей с БД должны завершить свои операции. Если по истечении указанного ниже периода времени остались активные соединения с пользователями, БД не переводится в однопользовательский режим, необходимый для создания резервной копии;
- **Deny new transactions while waiting** — существующие транзакции должны завершиться (Server Manager ожидает их завершения). Старт новых транзакций блокируется. Если по истечении указанного ниже периода времени остались активные транзакции, БД не переводится в однопользовательский режим;
- **Force shutdown after timeout** — активные транзакции могут выполняться в течение указанного ниже периода. Если в конце данного периода остаются активные транзакции,

они принудительно откатываются и затем все соединения с пользователями принудительно разрываются.

Резервное копирование базы данных

Для создания резервной копии БД следует выбрать команду **Database | Maintenance | Backup/Restore | Backup** и в диалоговом окне **Database Backup** указать путь к БД, которая должна служить источником для создания резервной копии, а также путь и имя файла резервной копии (рис. 7.19).

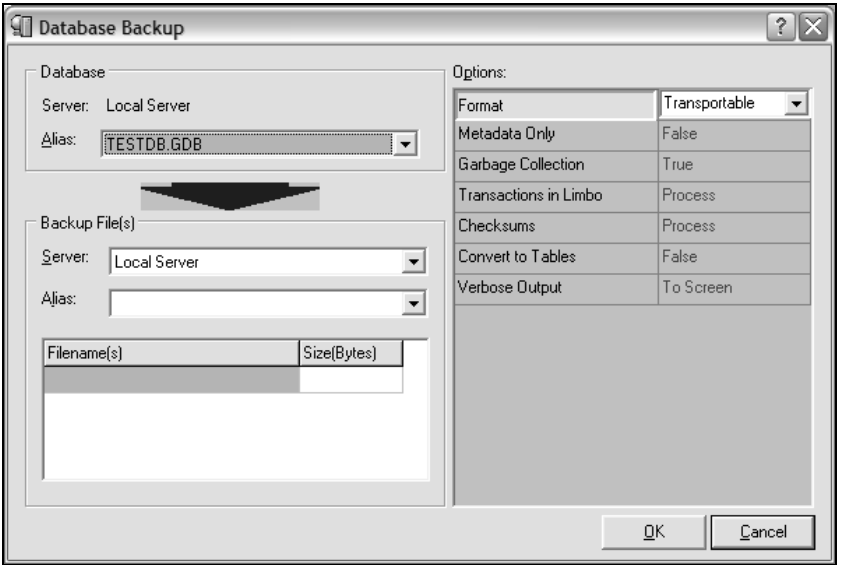


Рис. 7.19. Окно **Database Backup**

Кроме того, в разделе **Options** окна **Database Backup** можно указать необязательные параметры резервного копирования БД.

- **Format** — создание транспортного формата резервной копии в общем формате представления данных, что делает возможным перенесение БД в другие операционные среды,

для которых существуют реализации InterBase. Можно отключить эту опцию, выбрав **Non-Transportable**.

- **Metadata Only** — сохранять только метаданные, игнорируя страницы данных (режим **True**). Полезно в том случае, когда нужно создать заготовку пустой БД (например, при дистрибутировании разработанной программной системы).
- **Garbage Collection** — не производить/производить сборку мусора при сохранении БД.
- **Transactions in Limbo** — игнорировать либо не игнорировать при сохранении БД транзакции in limbo (транзакции, о которых нельзя сказать, завершились они или откатились; возникают при двухфазной фиксации транзакций для БД, расположенной на разных серверах, если операция подтверждена на одном сервере, а на другом в момент подтверждения транзакции произошел аппаратный сбой). Для восстановления транзакций in limbo перед созданием резервной копии следует выбрать команду **Database | Maintenance | Transaction Recovery**.
- **Checksums** — игнорировать неверные контрольные суммы заголовочных страниц БД, где хранится информация о БД, необходимая для соединения с нею; отключение данного режима (по умолчанию) предотвращает сохранение БД с разрушенной структурой.
- **Convert to Tables** — конвертация внутренних файлов базы данных во внешние таблицы, которые затем включаются как часть резерной копии.
- **Verbose Output** — получение расширенного отчета (в отдельном окне) о ходе создания резервной копии БД.

Восстановление базы данных из резервной копии

Для того чтобы восстановить БД из ранее сделанной резервной копии, необходимо выбрать в меню команду **Database | Maintenance | Backup/Restore | Restore**, а затем в диалоговом

окне указать путь и название резервной копии и БД, которая будет восстановлена. В том случае, если восстанавливаемая БД имеет несколько файлов либо располагается в нескольких физических файлах, в окне **Database Restore** (рис. 19.20) необходимо указать имя каждого файла БД и его размер в страницах в поле **Page Size**.

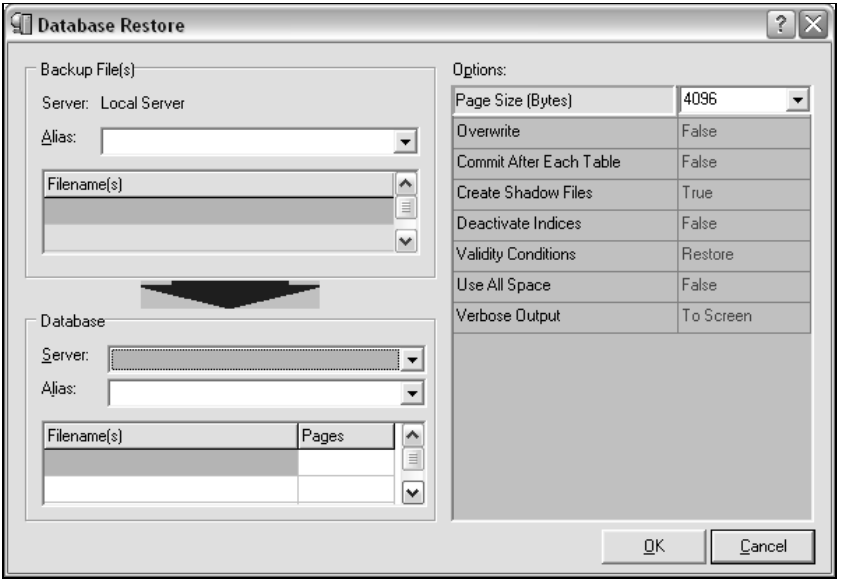


Рис. 7.20. Установка параметров для восстановления базы данных в окне **Database Restore**

В окне **Database Restore** могут быть установлены также следующие опции.

- **Page Size (Bytes)** — устанавливает размер страницы восстанавливаемой БД.
- **Overwrite** — заменяет существующую БД с тем же именем, если она расположена в том каталоге, куда должна помещаться восстанавливаемая БД.

- **Commit After Each Table** — подтверждать восстановление после каждой таблицы. Это актуально в тех случаях, когда в резервной копии имеются разрушенные данные и следует восстанавливать не все из них.
- **Create Shadow Files** — при восстановлении БД не создавать теневой (зеркальной) копии БД, если для этой БД назначено автоматическое ведение теневой копии. Это может быть необходимо в случаях:
  - восстановления БД на сервере, который не поддерживает теневого копирования;
  - восстанавливаемая БД сама является теневой копией БД.

  В восстановленной БД будут уничтожены при этом определения теневой БД, если восстановление БД, для которой ранее осуществлялось теневое копирование, осуществляется с включенным режимом **Create Shadow Files**.
- **Deactivate Indices** — отключить восстановление индексов. InterBase строит индексы после того, как БД восстановлена. Если в БД, служившей источником для создания резервной копии, существуют дублированные значения столбцов, по которым построены уникальные или первичные индексы, попытка перестройки индексов для восстановленной БД вызовет ошибку. В этом случае следует отключить восстановление индексов и затем, используя утилиту Interactive SQL, устранить дублирование значений. После этого можно активизировать индексы (SQL-оператор `ALTER INDEX`).
- **Validity Conditions** — восстанавливать или не восстанавливать определения ограничений ссылочной целостности. Это важно в том случае, когда резервная копия БД содержит устаревшие ограничения ссылочной целостности, которые должны быть заменены новыми.
- **Use All Space** — использовать все имеющееся пространство.

- **Verbose Output** — получение расширенного отчета (в отдельном окне, в файле, не делать отчета) о ходе восстановления БД из резервной копии.

Принудительная запись на диск

Записи таблиц при добавлении или изменении их в БД могут помещаться в буфер или немедленно физически записываться на диск. В первом случае записи из буфера физически записываются на диск после заполнения буфера. Режим накапливания изменений в буфере экономит время и улучшает быстродействие при работе с БД, поскольку достаточно медленное обращение к диску происходит реже. Однако в случае сбоя данные из буфера теряются, не будучи записаны на диск, что способно серьезно нарушить целостность БД. В силу этого рекомендуется метод немедленной (принудительной) записи на диск (forced writes). Отключить либо повторно включить его можно в диалоговом окне установки свойств БД командой **Database | Properties | General (Disabled** — запрещен, **Enabled** — разрешен).

Восстановление транзакций

В случае если транзакция затрагивает базы данных, расположенные на разных серверах, подтверждение такой транзакции осуществляется в две фазы. Все действия при этом выполняются автоматически и не требуют программирования. При возникновении ошибки в ходе выполнения двухфазной фиксации транзакция считается потерянной (in limbo), и относительно нее сервер БД не может решить, закончилась она или откатилась.

Для восстановления таких транзакций следует выбрать команду меню **Database | Maintenance | Transaction Recovery**. В диалоговом окне будут показаны в виде списка транзакции in limbo. Они могут быть подтверждены или отменены. Для ка-

ждой транзакции можно показать все связанные с ней транзакции (путем выбора знака "+" слева от транзакции).

Регистрация новых пользователей

Регистрация новых пользователей может осуществляться администратором системы. Для этого следует выбрать команду меню **Server | User Security**. В появившемся окне можно просмотреть список имен зарегистрированных на сервере пользователей (рис. 7.21).

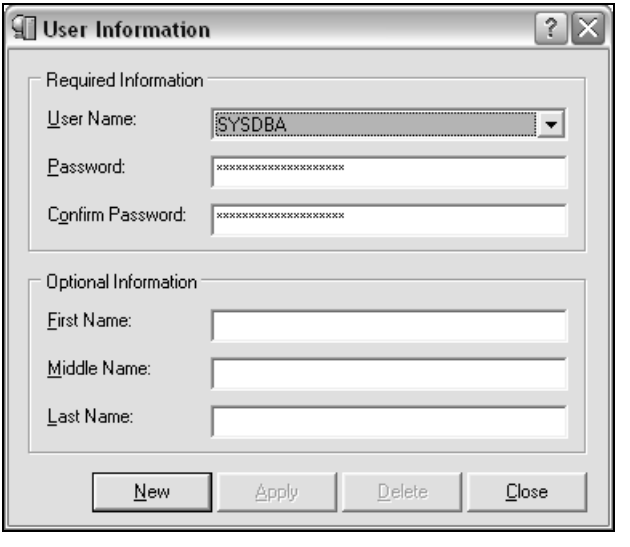


Рис. 7.21. Окно новых пользователей в окне **User Information**

Для добавления нового пользователя следует нажать кнопку **New** и указать имя пользователя и пароль, которые будут запрашиваться при доступе к базе данных. Для удаления пользователя следует в этом окне выбрать из списка удаляемого пользователя и нажать кнопку **Delete**.

Изменение данных, касающихся конкретного пользователя, производится следующим образом:

1. В левой части окна утилиты IBConsole для данного сервера выделяется **Users** (рис. 7.22).
2. В правой части среди появившихся пользователей следует вызвать (нажатием правой кнопки мыши) контекстное меню для изменения параметров существующего пользователя.
3. Воспользоваться командой **Modify User**.

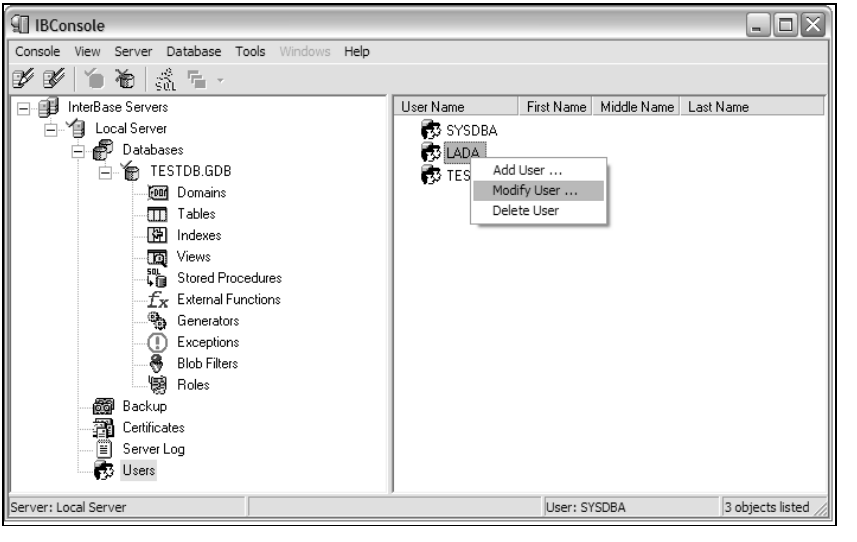


Рис. 7.22. Изменение данных пользователя в окне IBConsole

Работа с утилитой BDE Administrator

BDE Administrator предназначена для создания, изменения и удаления псевдонимов доступа к БД. С этой же целью может использоваться и утилита SQL Explorer.

Создание псевдонима базы данных

Параметры БД и ее местоположение определяются псевдонимом (именем, т. е. alias) БД. Псевдоним необходим для того, чтобы все изменения, связанные с базой данных, могли автоматически меняться без изменения созданных клиентских программ, в которых данный псевдоним используется.

Для определения псевдонима необходимо:

1. Выбрать команду меню **Object | New** в окне BDE Administrator (рис. 7.23).

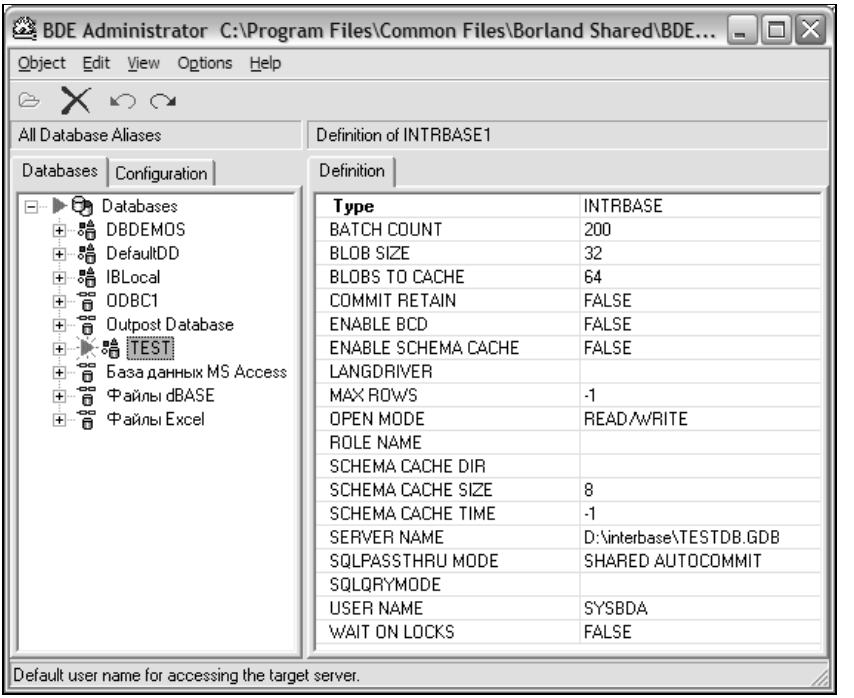


Рис. 7.23. Окно утилиты BDE Administrator

2. Определить имя драйвера базы данных (STANDARD для БД, созданных с помощью Paradox, dBase, Clipper и FoxPro. MSACCESS для Microsoft Access. ORACLE, INTRBASE, SYBASE, MSSQL, INFORMIX, DB2 соответственно для баз данных Oracle, InterBase, Sybase, MS SQL Server, Informix, DB2 и, если установлен, драйвер ODBC).
3. Ввести имя псевдонима в левой части окна BDE Administrator (рис. 7.23).
4. Определить необходимые параметры псевдонима в правой части окна BDE Administrator.
5. Из контекстного меню созданного псевдонима (правая кнопка мыши) выбрать команду **Apply** для подтверждения его создания.

Создание псевдонима *INTRBASE*

Псевдонимы типа `INTRBASE` используются для доступа к базе данных сервера InterBase. Параметры, которые необходимо определить для псевдонима типа `INTERBASE`, приведены в табл. 7.3.

Таблица 7.3. *Параметры для определения псевдонима типа* `INTERBASE`

Параметр	Описание
`BATCH COUNT`	Число записей, изменяемых в рамках неявной транзакции. Перед изменением записей автоматически стартует транзакция на сервере. После изменения этого числа записей транзакция на сервере автоматически подтверждается, Использование `BATCH COUNT` дает возможность изменить число записей в пакете обновлений для каждой подтвержденной транзакции, если размер подобного пакета на сервере недостаточно велик. По умолчанию — число записей, умещающихся в пакете длиной 32 Кбайт

Таблица 7.3 *(продолжение)*

Параметр	Описание
BLOB SIZE	Определяет максимальную длину (в килобайтах) BLOB-попя. Может меняться от 32 до 1000
BLOBS TO CACHE	Определяет, сколько записей с полями BLOB будут кэшироваться для клиента. По умолчанию 64. Может меняться от 64 до 65 535
ENABLE BCD	Указывает на необходимость для BOE переводить целые и десятичные значения полей в значения BCD
ENABLE SCHEMA CACHE	При значении TRUE позволяет хранить сведения о структуре удаленной БД на клиентском компьютере в каталоге, определяемом параметром SCHEMA CACHE DIR. Хранение структуры БД хорошо тем, что в этом случае BDE не нужно всякий раз считывать данную информацию из удаленной БД, что экономит время и ресурсы. Однако это возможно лишь для БД, чья структура (состав таблиц, столбцов, индексов и др.) не изменяется во времени
SCHEMA CACHE TIME	В случае хранения сведений о структуре БД параметр указывает, с какой периодичностью BDE обновляет сведения о ней. Значения: 0 — хранение схемы БД не используется; (по умолчанию) — после закрытия БД в приложении (т. е. после разрыва соединения с БД); число в диапазоне 1..2 147 483 647 — количество секунд между двумя обновлениями сведений о структуре БД
LANGDRIVER	Языковый драйвер, используемый для кодировки символьных полей и определяющий также порядок сортировки символьных значений. Для русскоязычных приложений рекомендуется Pdox ANSI Cyrillic. Для этого драйвера без ошибок работают символьные функции преобразования строчных/заглавных букв AnsiUpperCase и AnsiLowerCase.

Таблица 7.3 *(продолжение)*

Параметр	Описание
	Pdox ANSI Cyrillic совместим с кодировкой символов InterBase WIN1251 и порядком сортировки символов PXW_CYRL. Эти значения указываются в БД в атрибутах: `DEFAULT CHARACTER SET` для баз данных, `CHARACTER SET` и `COLLATE` для доменов и столбцов
`MAX ROWS`	Указывает максимальное количество записей, которые драйвер пытается считать из удаленной БД при посылке каждого SQL-запроса к серверу, с учетом запросов на чтение структуры БД и проверки соответствия значений столбцов накладываемым на них ограничениям. Используется для предотвращения безграничного расхода системных ресурсов, например, при выдаче оператора `SELECT` для большой по объему таблицы БД. Малое значение MAX ROWS может привести к тому, что таблица БД не будет открыта, поскольку запрос чтения структуры таблицы может возвратить больше записей, чем указано в MAX ROWS. Однако MAX ROWS никак не воздействует на необновляемые запросы. По умолчанию принято –1, что означает отсутствие лимита на количество считываемых строк. В случае если количество возвращаемых строк больше установленного лимита, выдается ошибка `DBIERR_ROWFETCHLIMIT`
`OPEN MODE`	Устанавливает режим чтения (записи) данных из (в) БД (значение `READ/WRITE`, по умолчанию) или режим только для чтения (`READ ONLY`)
`SERVER NAME`	Для удаленных БД содержит имя сетевого сервера и маршрут доступа к БД. Вид указания этих параметров зависит от используемого сетевого протокола. Для протокола TCP/IP при расположении БД на сервере, который работает под управлением Windows, они задаются в формате: **ИмяСервера: Путь\ИмяБД.gdb** например: abcd:d:\database\db_1.gdb

Таблица 7.3 *(продолжение)*

Параметр	Описание
	При этом должны быть соответствующим образом настроены файлы HOSTS и SERVICES (располагаются в каталоге Windows) на компьютере, имеющем доступ к удаленной БД. В файле HOSTS описываются IP-адрес сервера и имя, например: 192.168.92.167 grd В файле SERVICES — протокол доступа к InterBase: gds_db 3050/tcp
`SQLPASSTHRU MODE`	Режим технологии "клиент-сервер", определяющий, каким образом происходит взаимодействие BDE с сервером на уровне транзакций приложения клиента. Значение параметра определяет, может ли BDE передавать серверу собственные команды для управления запросами, или нет. В последнем случае используется Passthrough SQL — операторы SQL, выполняемые с помощью компонента `TQuery` клиентского приложения. Также данный параметр определяет режимы использования неявного старта и подтверждения транзакций. Параметр `SQLPASSTHRU MODE` может принимать следующие значения: ❑ `SHARED AUTOCOMMIT` — Passthrough SQL и команды BDE используют одно и то же соединение с удаленной БД, реализуемое в приложении через один компонент `TDatabase`; если в приложении транзакция явно не реализуется методом `TDatabase.StartTransaction`, происходит неявный старт транзакции и ее автоматическое подтверждение; ❑ `SHARED NOAUTOCOMMIT` — Passthrough SQL и команды BDE используют одно и то же соединение с удаленной БД, неявные транзакции стартуют аналогично режиму `SHARED AUTOCOMMIT`, однако их неявного подтверждения не происходит и это нужно делать явно; поведение Passthrough SQL в этом случае зависит от сервера;

Таблица 7.3 *(окончание)*

Параметр	Описание
	❑ NOT SHARED — Passthrough SQL и команды BDE не могут использовать одно и то же соединение с удаленной БД. Обновляемые запросы на SQL не поддерживаются псевдонимами БД, для которых установлен этот режим. SHARED AUTOCOMMIT и SHARED NOAUTOCOMMIT не поддерживают всех операторов Passthrough SQL. В этих режимах не следует управлять транзакциями с помощью SQL-оператора SET TRANSACTION
SQLQRYMODE	Определяет режим выполнения запросов SQL. Возможные значения: ❑ NULL (по умолчанию) — для доступа как к локальным, так и серверным БД: запросы вначале посылаются SQL-серверу, если тот не в состоянии выполнить запрос, он выполняется локально; ❑ SERVER — только серверные БД: запрос посылается SQL-серверу, если он не может выполнить запрос, локального выполнения не происходит; ❑ LOCAL — запрос всегда выполняется на клиентской машине, т. е. локально. Заметим, что запрос может быть отвергнут сервером БД, если: • имеет место гетерогенный запрос, т. е. запрос к различным БД; • запрос состоит более чем из одного SQL-оператора; • сервер БД не поддерживает синтаксис операторов, присутствующих в запросе
USER NAME	Определяет имя, которое выдается в качестве подсказки имени пользователя при запросе имени и пароля в момент соединения с БД. При определении следует изначально вписать SYSDBA

Установки параметров драйвера

Параметры, которые характеризуют все базы данных определенного типа, устанавливаются для драйверов БД. Затем они применяются к псевдониму каждой БД того же типа. Некоторые (не все) параметры могут быть переопределены для конкретного псевдонима.

Установка параметров драйверов производится на вкладке **Configuration** в разделе **Drivers | Native | Имя драйвера** (рис. 7.24).

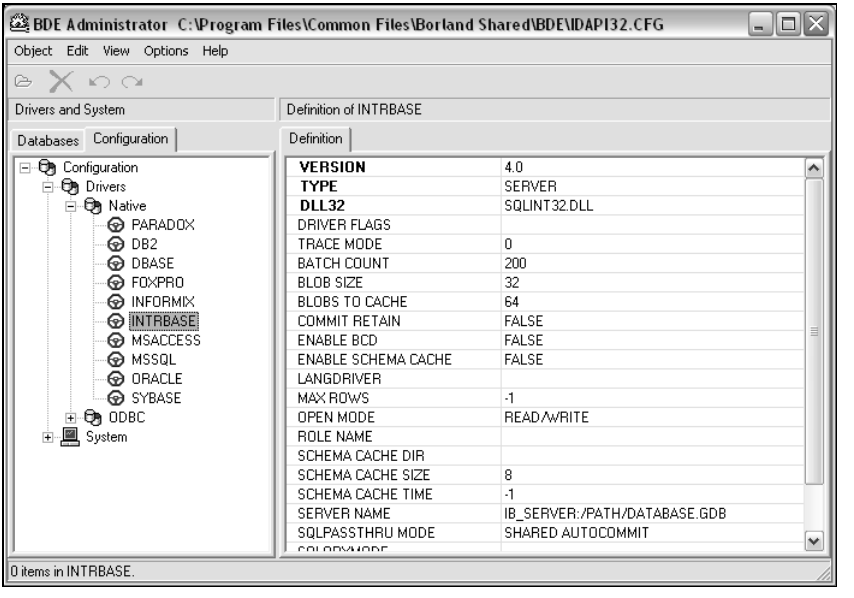


Рис. 7.24. Установка необходимых параметров драйвера

Системные стартовые установки

Системные стартовые установки можно изменить на вкладке **Configuration** в разделе **System | INIT** (рис. 7.25). Здесь можно установить стартовые параметры запуска приложения, рабо-

тающего с BDE. Как правило, установки языкового драйвера меняются редко и при работе с BDE не рекомендуется вносить изменения в данные, которые приняты по умолчанию.

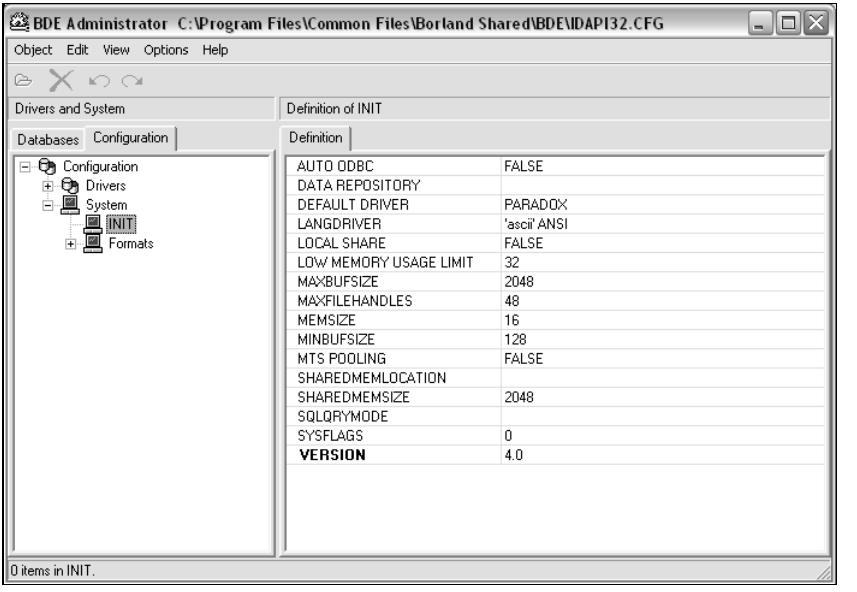


Рис. 7.25. Окно BDE Administrator для изменения системных установок

Установки форматов

Установки форматов (даты — **Date**, времени — **Time**, числовых значений — **Number**) изменяются также на вкладке **Configuration** в разделе **System | Formats** (рис. 7.26).

Сохранение конфигурации

Текущая конфигурация со значениями псевдонимов, драйверов и стартовых настроек может быть сохранена в файле. Для этого необходимо выбрать команду меню **Object | Save As Configura-**

tion, затем указать имя файла и нажать кнопку **OK**. По умолчанию конфигурация записывается в файл C:\Program Files\Borland\Common Files\BDE\Idapi32.cfg. Данный файл обновляется автоматически после выполнения подтверждения для всех изменений **(Object | Apply)** или для каждого изменения **(Apply** в контекстном меню псевдонима). Файл конфигурации, отличный от принятого по умолчанию, может быть загружен с помощью команды **Object | Open Configuration**.

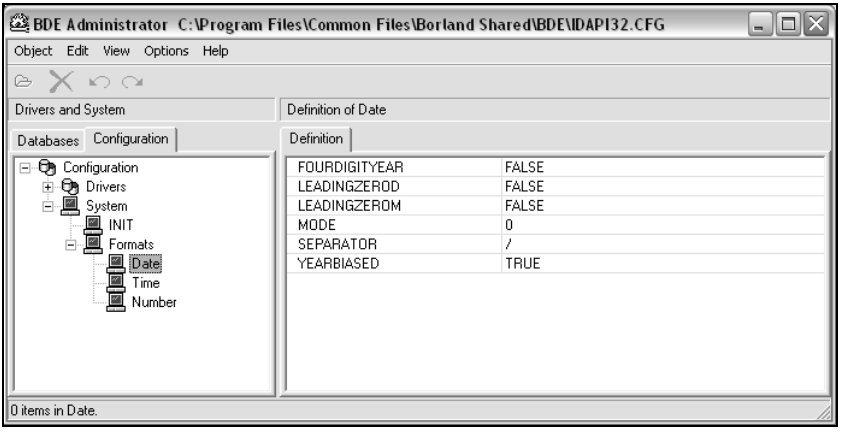


Рис. 7.26. Окно BDE Administrator для изменения форматов даты, времени и числовых значений

Файлы конфигурации можно объединять, когда необходимо установить на компьютер приложение или группу приложений и добавить в файл конфигурации новые псевдонимы баз данных. Если вместе с дистрибутивом приложений поставляется и файл конфигурации, то псевдонимы из него могут быть добавлены в основную (или любую другую) конфигурацию. Для этого следует выбрать команду меню **Object | Merge Configuration** и указать конфигурационный файл, информация из которого будет добавляться, после чего подтвердить объединение **(Yes)**.

Пример разработки клиентского приложения с использованием InterBase и Delphi

Приведем постановку задачи.

Разработать и реализовать клиентское приложение, позволяющее работать с базой данных "Музыкальные произведения". В базе данных должна храниться следующая информация: фамилия, имя и отчество композитора, год рождения, страна происхождения композитора. Информация о произведении содержит год написания, название, темп музыки, основную тональность (соль минор, соль мажор, до минор, до мажор и др.). Кроме того, выделяют различные формы произведения: симфония, этюд, вальс, соната, опера, балет и т. д.

Проектирование базы данных

Исходя из поставленной задачи, можно определить следующие сущности, отражающие основные объекты предметной области.

Сущность `Список композиторов` характеризуется следующими атрибутами: `Ф.И.О композитора`; `Год рождения`; `Страна`.

Сущность `Список произведений` имеет атрибуты: `Год написания`; `Название`; `Темп`; `Основная тональность` (соль минор, соль мажор и т. д.).

Для сущности `Формы произведений` можно определить атрибут `Форма произведения` (опера, балет, оперетта, пьеса и т. д.).

Кроме того, каждый композитор характеризуется идентификатором, отличающим его уникальным образом в базе данных. Каждая форма произведения также имеет уникальный идентификатор.

Каждый композитор может написать то либо иное произведение, но каждое произведение обязано иметь автора. Аналогично, каждая форма произведения может характеризовать

конкретное музыкальное произведение, и каждое музыкальное произведение обязано принадлежать той либо иной форме произведения.

Как правило, в настоящее время генерацию SQL-кода для базы данных можно проводить с использованием того либо иного CASE-средства (Computer-Aided Software / System Engineering), которое позволяет с помощью определенных графических нотаций строить концептуальную модель базы данных и получать SQL-скрипт для конкретного сервера базы данных.

С учетом всего изложенного концептуальная модель для базы данных "Музыкальные произведения" может быть представлена в CASE-средстве PowerDesigner в соответствии с рис. 7.27. При определении основных элементов концептуальной диаграммы были установлены также три домена для темпа, тональности и формы музыкального произведения.

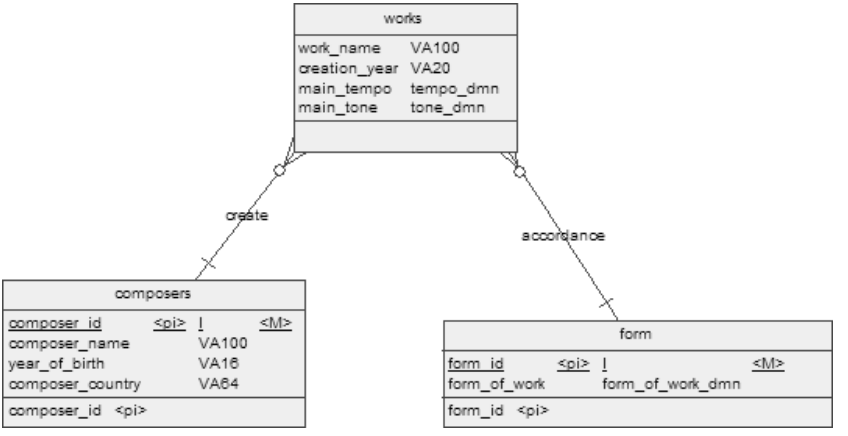


Рис. 7.27. Концептуальная диаграмма для базы данных "Музыкальные произведения"

Итак, полученная концептуальная модель является формализованным представлением общей структуры данных информационной системы без привязки к конкретной реализации на

базе СУБД. Для получения физической модели данных необходимо выбрать соответствующую СУБД. В данном примере генерация физической модели данных производится для сервера баз данных InterBase.

Перед генерацией физической модели данных полагаем, что ограничения будут заданы внутри схемы — при установке соответствующих опций в PowerDesigner следует выбрать **Declarative**.

Физическая модель для базы данных "Музыкальные произведения" приведена на рис. 7.28.

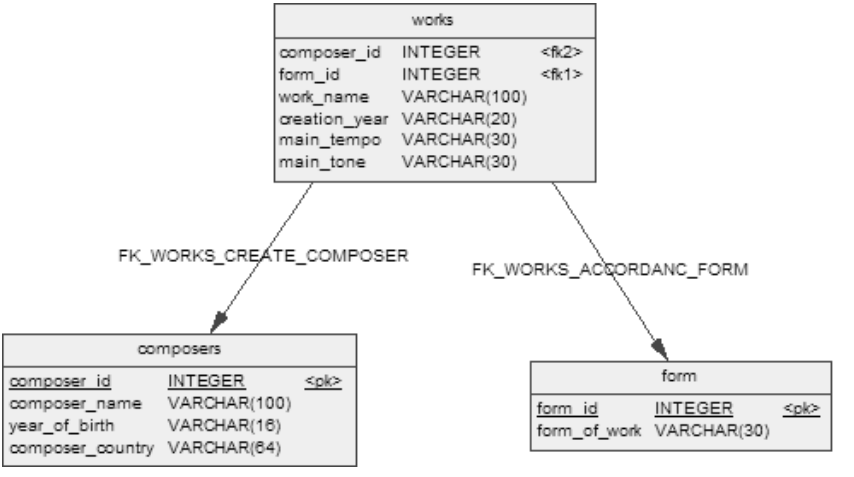


Рис. 7.28. Физическая модель для базы данных "Музыкальные произведения"

Генерация SQL-скрипта

Для генерации SQL-скрипта на основе полученной физической модели данных используется команда **DataBase | Generate DataBase** (линейки меню PowerDesigner).

В окне **Database Generation** (рис. 7.29) необходимо указать: в поле **DBMS** имя СУБД, для которой создается скрипт; в поле **Directory** — размещение на диске создаваемого файла, содержащего SQL-скрипт; в поле **File name** — имя файла с расширением sql. Значение **Generation type** должно быть установлено в **Script generation**.

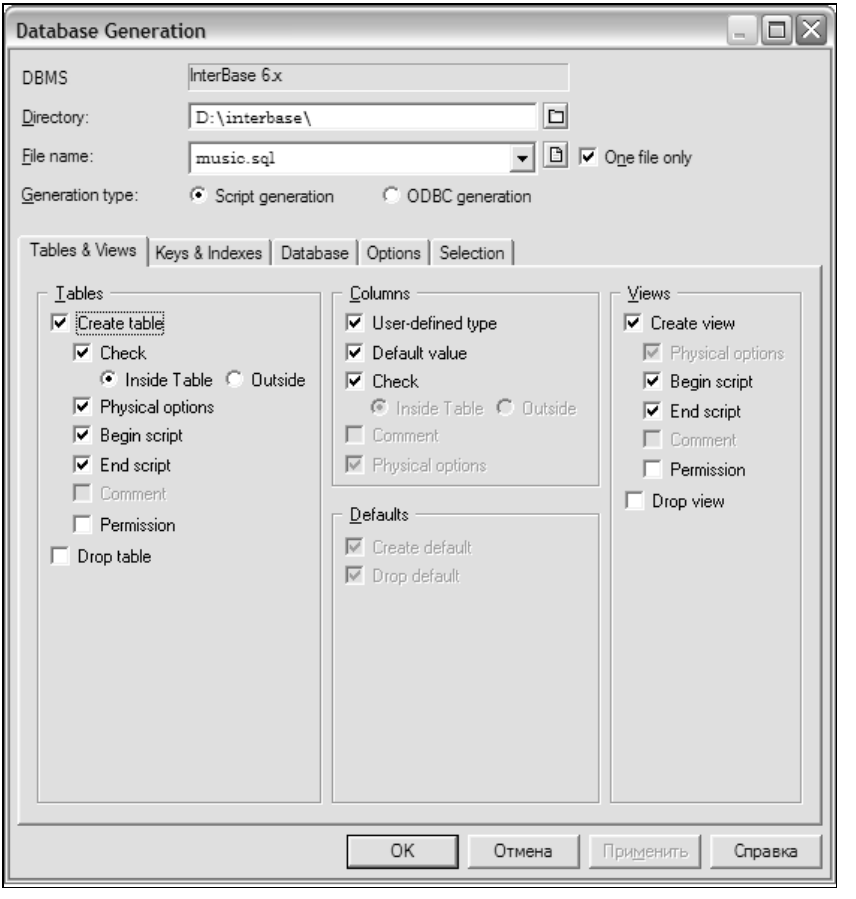


Рис. 7.29. Окно **Database Generation**

Кроме того, для успешной генерации SQL-скрипта следует на всех вкладках окна **Database Generation** установить флажки в соответствии с рис. 7.30—7.32.

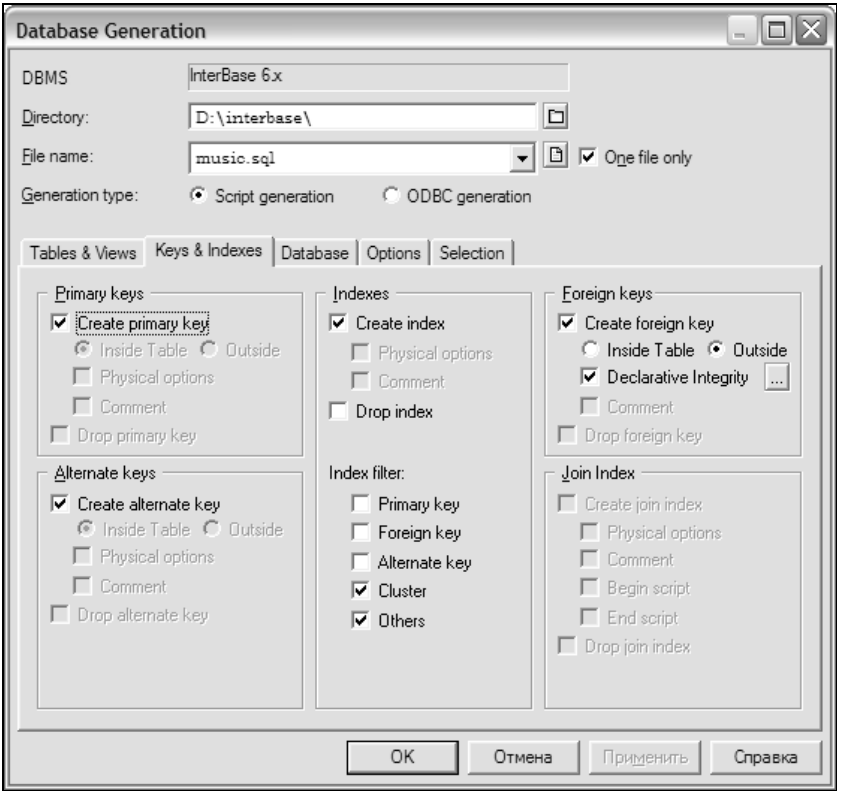


Рис. 7.30. Окно **Database Generation** — вкладка **Keys & Indexes**

После установки всех необходимых опций следует подтвердить генерацию SQL-скрипта нажатием кнопки **OK**. В результате появляются два окна, содержащие информацию о произошедшей генерации: **Result List** и **Result**. Окно **Result List**

содержит информацию об ошибках, произошедших во время генерации, а также предупреждения о возможности возникновения ошибок при использовании этого скрипта для создания базы. Окно **Result** отображает полные имена созданных во время генерации файлов. В случае если генерация прошла успешно, в указанной папке на диске создается файл с заданным именем, содержащий необходимый SQL-скрипт. В данном примере получается файл music.sql, который приведен в листинге 7.1.

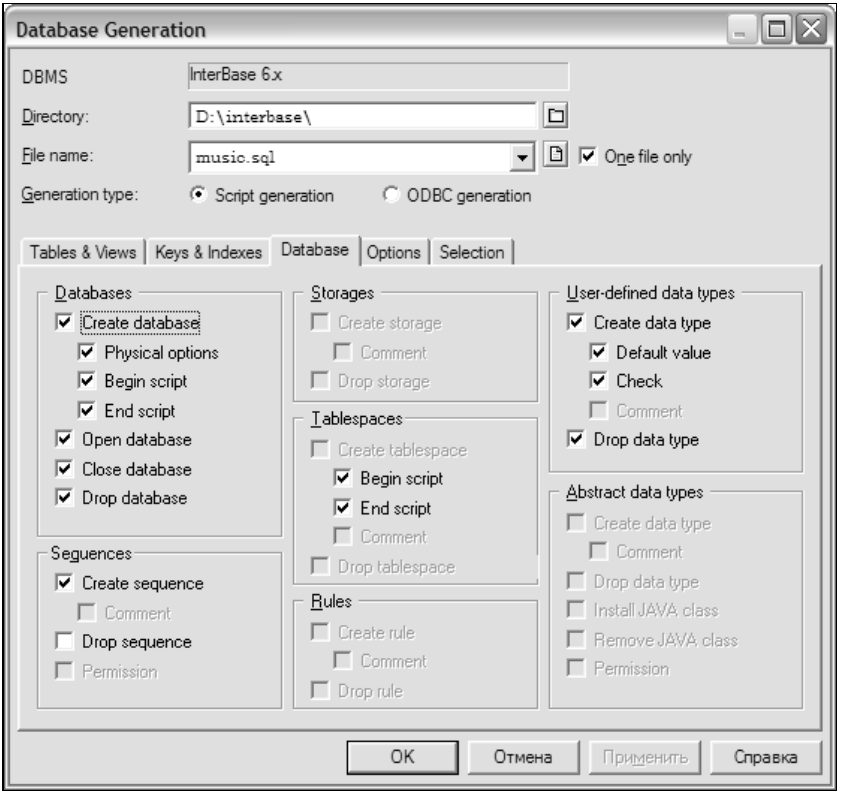


Рис. 7.31. Окно **Database Generation** — вкладка **Database**

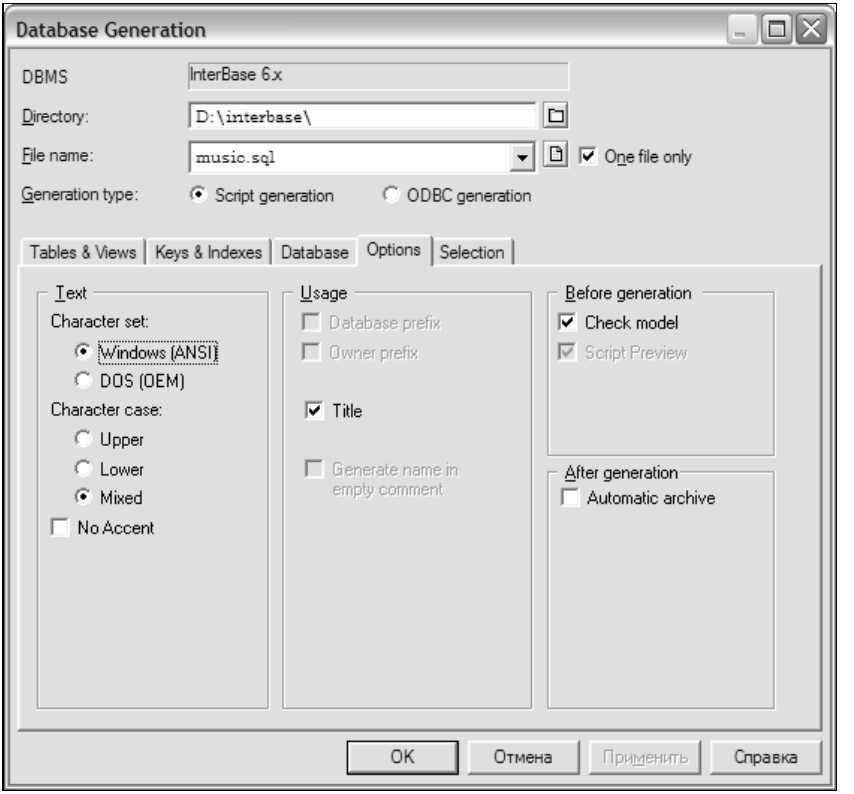


Рис. 7.32. Окно **Database Generation** — вкладка **Options**

Листинг 7.1. Файл music.sql

```
CREATE DOMAIN "FORM_OF_WORK_DMN" AS VARCHAR(30) CHARACTER SET
WIN1251
 check (value is null or ( value in
('symphony','etude','waltz','sonata','sonatina','fugue','oper
a','operetta','ballet','other') ));
CREATE DOMAIN "TEMPO_DMN" AS VARCHAR(30) CHARACTER SET
WIN1251
```

```
 check (value is null or ( value in
('allegro','allegretto','moderato','andante','andantino','len
to','grave','presto','prestissimo','largo','adagio') ));
CREATE DOMAIN "TONE_DMN" AS VARCHAR(30) CHARACTER SET WIN1251
 check (value is null or ( value in ('c dur','c moll','d
dur','d moll','e dur','e moll','f dur','f moll','g dur','g
moll','a dur','a moll','h dur','h moll') ));


CREATE TABLE "COMPOSERS"
(
  "COMPOSER_ID"INTEGER NOT NULL,
  "COMPOSER_NAME"VARCHAR(100) CHARACTER SET WIN1251,
  "YEAR_OF_BIRTH"VARCHAR(16) CHARACTER SET WIN1251,
  "COMPOSER_COUNTRY"VARCHAR(64) CHARACTER SET WIN1251,
CONSTRAINT "PK_COMPOSERS" PRIMARY KEY ("COMPOSER_ID")
);


CREATE TABLE "FORM"
(
  "FORM_ID"INTEGER NOT NULL,
  "FORM_OF_WORK""FORM_OF_WORK_DMN",
CONSTRAINT "PK_FORM" PRIMARY KEY ("FORM_ID")
);


CREATE TABLE "WORKS"
(
  "COMPOSER_ID"INTEGER NOT NULL,
  "FORM_ID"INTEGER NOT NULL,
  "WORK_NAME"VARCHAR(100) CHARACTER SET WIN1251,
  "CREATION_YEAR"VARCHAR(20) CHARACTER SET WIN1251,
  "MAIN_TEMPO""TEMPO_DMN",
  "MAIN_TONE""TONE_DMN"
);
```

```
ALTER TABLE "WORKS" ADD CONSTRAINT "FK_WORKS_ACCORDANC_FORM"
FOREIGN KEY ("FORM_ID") REFERENCES "FORM" ("FORM_ID");

ALTER TABLE "WORKS" ADD CONSTRAINT "FK_WORKS_CREATE_COMPOSER"
FOREIGN KEY ("COMPOSER_ID") REFERENCES "COMPOSERS"
("COMPOSER_ID");


ALTER TABLE "FORM" ADD

CONSTRAINT "CKC_FORM_OF_WORK_FORM"

check (form_of_work is null or ( form_of_work in
('symphony','etude','waltz','sonata','sonatina','fugue','oper
a','operetta','ballet','other') ));


ALTER TABLE "WORKS" ADD

CONSTRAINT "CKC_MAIN_TONE_WORKS"

check (main_tone is null or ( main_tone in ('c dur','c
moll','d dur','d moll','e dur','e moll','f dur','f moll','g
dur','g moll','a dur','a moll','h dur','h moll') ));
```

Полученный SQL-скрипт будет в дальнейшем использоваться для создания базы данных.

Как отмечалось ранее, в SQL-скрипте из листинга 7.1 ограничения заданы внутри схемы базы данных. Для реализации ограничений в виде триггеров, а также генераторов случайных чисел и хранимых процедур для присвоения уникальных имен следует использовать листинг 7.2, который получается при задании соответствующих опций в PowerDesigner для генерации физической модели и SQL-скрипта.

Листинг 7.2. Файл music.sql (учетены ограничения в виде триггеров, а также – генераторы случайных чисел и хранимые процедуры)

```
/* Создание домена "FORM_OF_WORK_DMN" для определения формы
произведения */


CREATE DOMAIN "FORM_OF_WORK_DMN" AS VARCHAR(30) CHARACTER SET
WIN1251
```

```
 check (value is null or ( value in
('symphony','etude','waltz','sonata','sonatina','fugue','oper
a','operetta','ballet','other') ));


/* Создание домена "TEMPO_DMN" для определения темпа
произведения */


CREATE DOMAIN "TEMPO_DMN" AS VARCHAR(30) CHARACTER SET
WIN1251
 check (value is null or ( value in
('allegro','allegretto','moderato','andante','andantino','len
to','grave','presto','prestissimo','largo','adagio') ));


/* Создание домена "TONE_DMN" для определения тональности
произведения */


CREATE DOMAIN "TONE_DMN" AS VARCHAR(30) CHARACTER SET WIN1251
 check (value is null or ( value in ('c dur','c moll','d
dur','d moll','e dur','e moll','f dur','f moll','g dur','g
moll','a dur','a moll','h dur','h moll') ));



/* Создание таблицы "COMPOSERS" для хранения данных
о композиторах */


CREATE TABLE "COMPOSERS"
(
  "COMPOSER_ID"INTEGER NOT NULL,
  "COMPOSER_NAME"VARCHAR(100) CHARACTER SET WIN1251,
  "YEAR_OF_BIRTH"VARCHAR(16) CHARACTER SET WIN1251,
  "COMPOSER_COUNTRY"VARCHAR(64) CHARACTER SET WIN1251,
CONSTRAINT "PK_COMPOSERS" PRIMARY KEY ("COMPOSER_ID")
);



/* Создание таблицы "FORM" для хранения данных
о форме произведений */
```

```
CREATE TABLE "FORM"
(
  "FORM_ID"INTEGER NOT NULL,
  "FORM_OF_WORK""FORM_OF_WORK_DMN",
CONSTRAINT "PK_FORM" PRIMARY KEY ("FORM_ID")
);



/* Создание таблицы "WORKS" для хранения данных
о произведениях */

CREATE TABLE "WORKS"
(
  "COMPOSER_ID"INTEGER NOT NULL,
  "FORM_ID"INTEGER NOT NULL,
  "WORK_NAME"VARCHAR(100) CHARACTER SET WIN1251,
  "CREATION_YEAR"VARCHAR(20) CHARACTER SET WIN1251,
  "MAIN_TEMPO""TEMPO_DMN",
  "MAIN_TONE""TONE_DMN"
);



/* Создание триггера для каскадного обновления данных
в таблице "WORKS" в случае, если произошло обновление данных
в таблице "COMPOSERS" */

create trigger before_update_COMPOSER_ID for COMPOSERS
active before update
as
begin
        if  (old.COMPOSER_ID <> new.COMPOSER_ID) then update
WORKS
         set  COMPOSER_ID=new.COMPOSER_ID
          where  COMPOSER_ID=old.COMPOSER_ID;
end;
```

```
/* Создание триггера для каскадного удаления данных
из таблицы WORKS в случае, если произошло удаление  данных
в таблице COMPOSERS */

create trigger after_delete_COMPOSER_ID for COMPOSERS
active after delete
as
begin
 delete from WORKS
   where WORKS.COMPOSER_ID=COMPOSERS.COMPOSER_ID;
end;



/* Создание триггера для каскадного обновления данных
в таблице WORKS в случае, если произошло обновление данных
в таблице FORM */



create trigger before_update_FORM_ID for FORM
active before update
as
begin
     if  (old.FORM_ID<> new.FORM_ID) then update WORKS
         set  FORM_ID=new.FORM_ID
          where  FORM_ID=old.FORM_ID;
end;



/* Создание триггера для каскадного удаления данных
из таблицы WORKS в случае, если произошло удаление данных
в таблице FORM */

create trigger after_delete_FORM_ID for FORM
active after delete
as
```

```
begin
  delete from WORKS
   where WORKS.FORM_ID=FORM.FORM_ID;
end;



/* Создание генератора, обеспечивающего уникальные значения
для столбца FORM_ID таблицы FORM */

create generator FORM_FORM_ID;
   set generator FORM_FORM_ID to 1;



/* Создание процедуры, в которой происходит обращение
к генератору для получения уникальных значений столбца
FORM_ID таблицы FORM */

   create procedure get_FORM_ID
   returns(n_id integer)
   as
   begin
        n_id=gen_id(FORM_FORM_ID,1);
   end;

/* Создание генератора, обеспечивающего уникальные значения
для столбца COMPOSER_ID таблицы COMPOSERS */

create generator COMPOSERS_COMPOSER_ID;
   set generator COMPOSERS_COMPOSER_ID to 1;



/* Создание процедуры, в которой происходит обращение
к генератору для получения уникальных значений столбца
COMPOSER_ID таблицы COMPOSERS */
```

```
create procedure get_COMPOSER_ID
returns(n_id integer)
as
begin
     n_id=gen_id(COMPOSERS_COMPOSER_ID,1);
end;
```

Создание базы данных с помощью утилиты IBConsole

При создании двухзвенных приложений следует помнить, что сервер базы данных размещается на некотором компьютере в сети, к которому будут обращаться клиентские приложения. Таким образом, удаленный сервер создается как локальный сервер на том компьютере сети, который выбран в качестве сервера баз данных. Клиентские приложения размещаются на локальных компьютерах сети. При этом на локальных компьютерах необходимо обязательно зарегистрировать удаленную базу данных с использованием утилиты BDE Administrator (либо SQL Explorer).

Приведем пример создания удаленной базы данных и приложения для работы с ней, находящегося на компьютере-сервере баз данных.

Прежде всего, необходимо зарегистрировать локальный сервер. Для этого можно воспользоваться командой **Server | Register** (выбрать в линейке меню окна утилиты IBConsole). В результате выполнения данной команды появится диалоговое окно **Register Server and Connect** (рис. 7.33).

В данном окне необходимо указать: тип сервера — локальный (**Local Server**); описание (в поле **Description**), если это необходимо; а также имя пользователя (в поле **User Name**) и пароль (в поле **Password**). Имя пользователя — SYSDBA, пароль — masterkey — в случае начала работы с утилитой; в дальнейшем можно изменить имя пользователя и пароль.

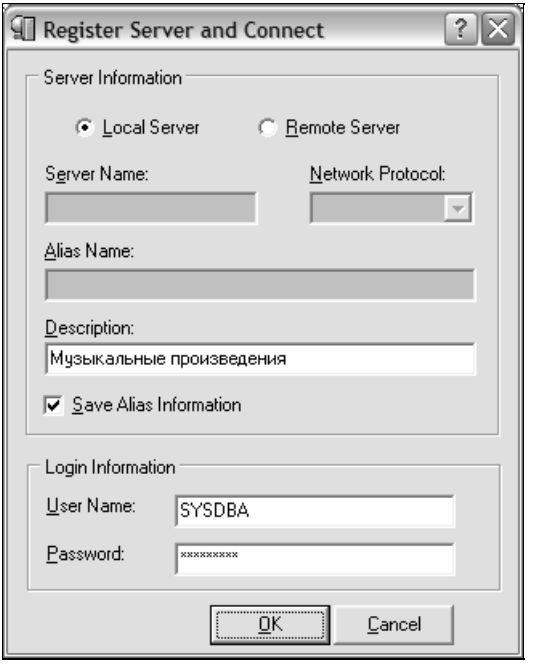


Рис. 7.33. Регистрация локального сервера базы данных

Теперь необходимо создать базу данных, для чего используется команда **Database | Create Database**. В окне **Create Database** (рис. 7.34) указывается псевдоним базы данных (поле **Alias**), прописывается путь нахождения файла базы данных в поле **Filename(s)** и указывается его первоначальный размер в поле **Size (Pages)**. Можно также указать некоторые дополнительные параметры в разделе **Options**:

- **Page Size** — размер для страницы хранения данных;
- **Default Character Set** — кодировка символов;
- **SQL Dialect** — диалект SQL.

После задания соответствующих опций в окне **Create Database** необходимо нажать кнопку **OK**.

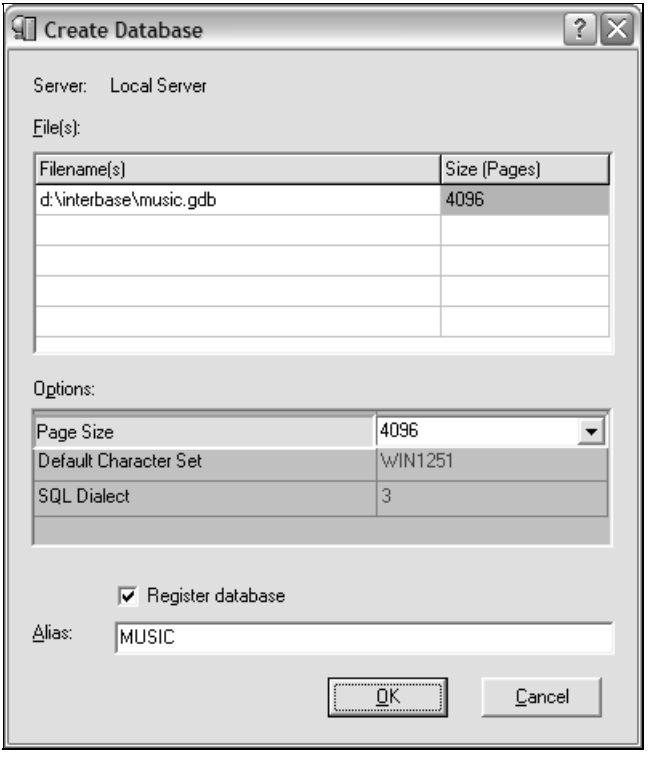


Рис. 7.34. Окно **Create Database**

Чтобы загрузить в базу созданный ранее SQL-скрипт, воспользуйтесь утилитой Interactive SQL (команда **Tools** | **Interactive SQL**).

Используя команду **Query** | **Load Script**, загружаем файл music.sql, содержащий SQL-скрипт. Он появится в рабочей области окна утилиты Interactive SQL (рис. 7.35).

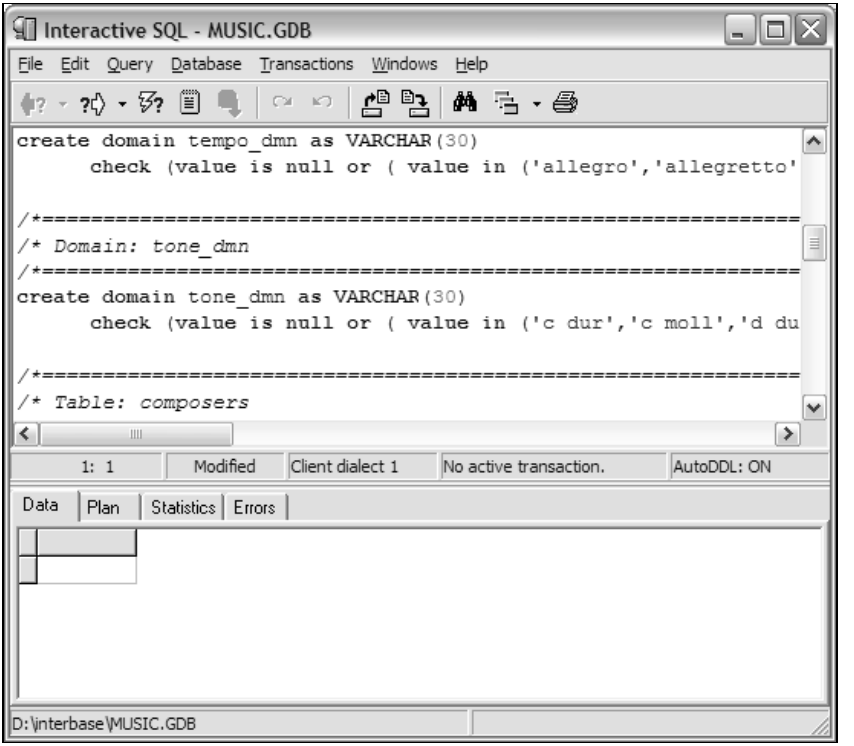


Рис. 7.35. Утилита Interactive SQL с SQL-скриптом для создания базы данных

Для выполнения скрипта используется команда линейки меню **Query** | **Execute** либо соответствующая кнопка панели инструментов. Если во время выполнения скрипта не произошло ошибок, рабочая область утилиты вновь станет пустой, т. е. необходимые элементы базы данных были успешно сгенерированы. Просмотр созданных объектов базы данных происходит в окне **IBConsole** (рис. 7.36): выделив соответствующий объект базы данных, расположенный в левой части окна, можно просмотреть и отредактировать любой из них.

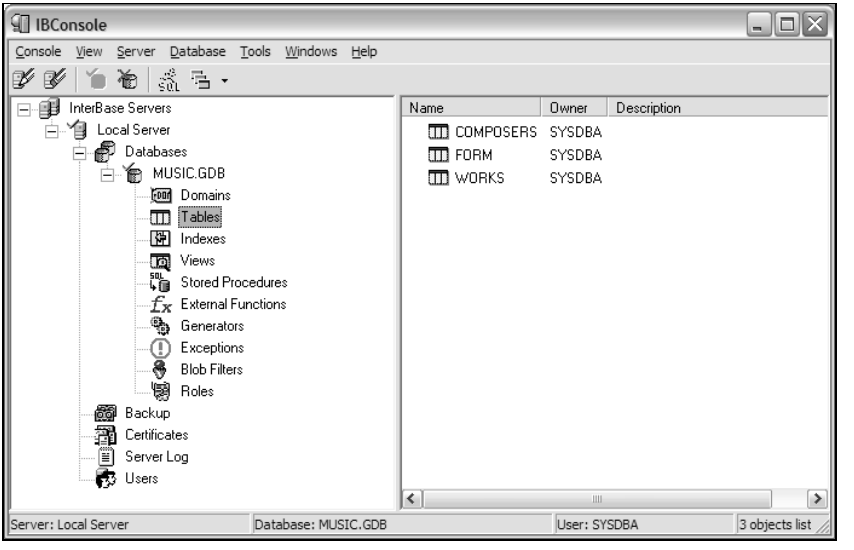


Рис. 7.36. Объекты базы данных "Музыкальные произведения" в окне **IBConsole**

Если во время выполнения SQL-скрипта произошли ошибки, то будет выдана развернутая информация, содержащая код ошибки: ее словесное описание, номер строки в файле, где произошла ошибка и т. д. Для успешного создания базы все ошибки в SQL-скрипте должны быть исправлены.

Утилита IBConsole позволяет также заполнять базу данных, модифицировать ее и получать необходимую информацию. Для просмотра возможностей работы с некоторой таблицей (например, добавление данных, поиск информации и т. п.), необходимо выбрать соответствующую таблицу и дважды щелкнуть по ней левой кнопкой мыши (либо воспользоваться контекстным меню таблицы, команда **Properties**), в результате чего откроется окно **Properties for: <имя выбранной таблицы>** (рис. 7.37).

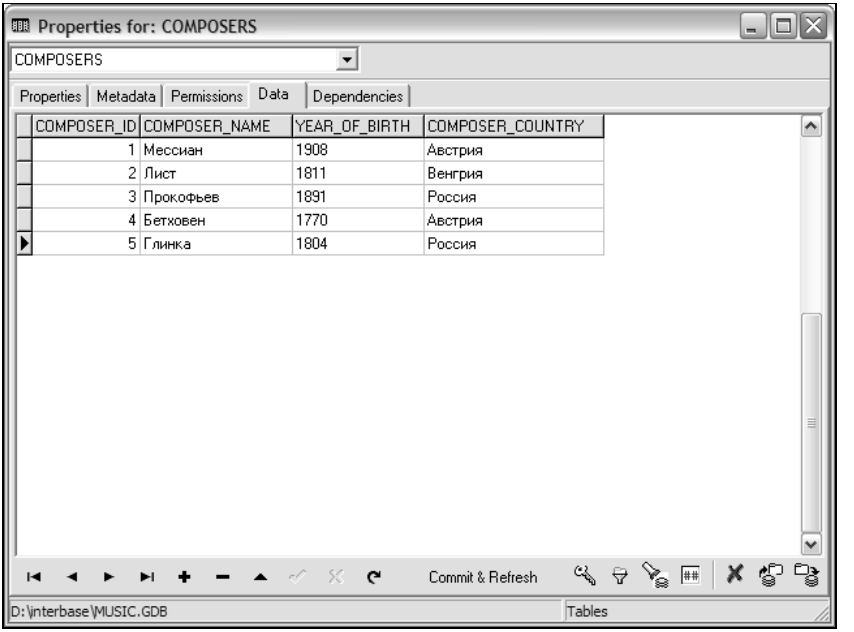


Рис. 7.37. Окно **Properties for: COMPOSERS**

В дальнейшем для работы с базой данных может быть использована утилита Interactive SQL. Однако для этого необходимо хорошо знать SQL-язык и принципы работы с утилитой Interactive SQL. Как правило, для пользователей базы данных создаются клиентские приложения с использованием, например, среды разработки Delphi, что упрощает приемы работы с информацией, находящейся в базе.

Разработка приложения в среде Delphi

Прежде чем приступить к программной реализации клиентского приложения, необходимо определить требования, предъявляемые к данной системе. Проанализировав постав-

ленную задачу, можно предположить, что система должна обладать следующими функциями:

- просмотр таблиц разработанной базы данных;
- возможность выполнения базовых действий над базой данных (добавление новых, а также удаление и редактирование уже имеющихся записей);
- выполнение всех видов SQL-запросов;
- возможность сохранения запросов в текстовых файлах с целью их повторного использования;
- понятный пользовательский интерфейс.

Кроме того, разрабатываемое приложение не должно быть зависимым от конкретной базы данных, т. е. должно служить некоторой универсальной оболочкой для работы с базами данных типа InterBase.

После выяснения необходимых требований будущей системы можно приступить к ее программной реализации.

Прежде всего, следует проверить, чтобы сервер баз данных InterBase был запущен и на нем размещена необходимая база данных (в данном случае music.gdb).

Далее, необходимо выбрать среду разработки приложений Delphi и создать новый проект, для чего следует воспользоваться командой линейки меню **File | New |Application** (рис. 7.38).

Так как приложение предназначено для работы с базами данных, то необходимо определить набор объектов, используемых при работе с ними. К таким объектам относятся: `BDatabase`; `IBTransaction`; `IBTable`; `IBQuery`; `DataSource`; `DBGrid`; `DBNavigator`.

Следует поместить данные компоненты на форму. Итак, компоненты `IBDatabase`, `IBTransaction`, `IBTable`, `IBQuery` расположены на вкладке **InterBase** (рис. 7.39).

Компонент `DataSource` — на вкладке **DataAccess** (рис. 7.40).

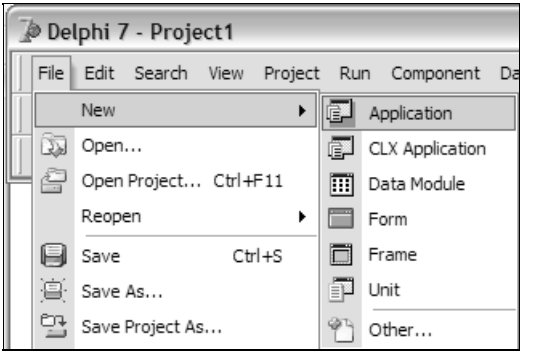


Рис. 7.38. Создание нового проекта в среде Delphi

Рис. 7.39. Компоненты вкладки **InterBase** панели инструментов **Component Palette**

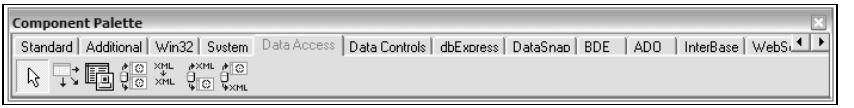


Рис. 7.40. Компоненты вкладки **DataAccess** панели инструментов **Component Palette**

Компоненты `DBGrid` и `DBNavigator` находятся на вкладке **DataControls** палитры компонентов (рис. 7.41).

Рис. 7.41. Компоненты вкладки **DataControls** панели инструментов **Component Palette**

Определим необходимые свойства для выбранных компонентов. Соединение с серверной базой данных осуществляется с помощью компонента `IBDatabase`. Для установки параметров соединения следует выполнить двойной щелчок мышью по компоненту `IBDatabase` и задать соответствующие опции (рис. 7.42).

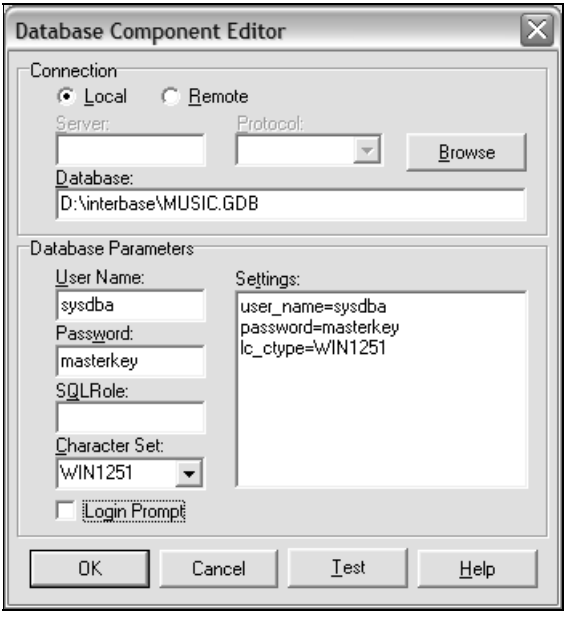


Рис. 7.42. Установка свойств соединения с базой данных

После того как параметры соединения заданы, определим также некоторые свойства компонента `IBDatabase`, для чего выделяем мышью данный компонент. При этом в инспекторе объектов (Object Inspector) отображаются свойства и значения выбранного компонента: значения некоторых из них необходимо изменить.

Прежде всего, можно изменить имя данного компонента, для этого его свойству `Name` присваивается соответствующее значение, например, `IBMyDBase` (псевдоним). Далее, в свойстве `DatabaseName` необходимо указать место размещения на диске файла базы данных (рис. 7.43).

Рис. 7.43. Установка свойств компонента `IBDatabase`

Кроме того, необходимо изменить значение свойства `DefaultTransaction` на `IBTransaction1`, которое отображается в выпадающем списке соответствующей строки инспектора объектов (см. также рис. 7.43).

Рассмотрим свойства компонента `IBTable`, который связан с активной таблицей базы данных. Значение следующих свойств данного компонента необходимо установить в соответствии с рис. 7.44:

- `Name` — присвоить значение `IBActiveTable`;
- `Database` — присвоить значение `IBMyDBase` (псевдоним используемой базы данных);
- `IBTransaction` — присвоить значение `Transaction1`.

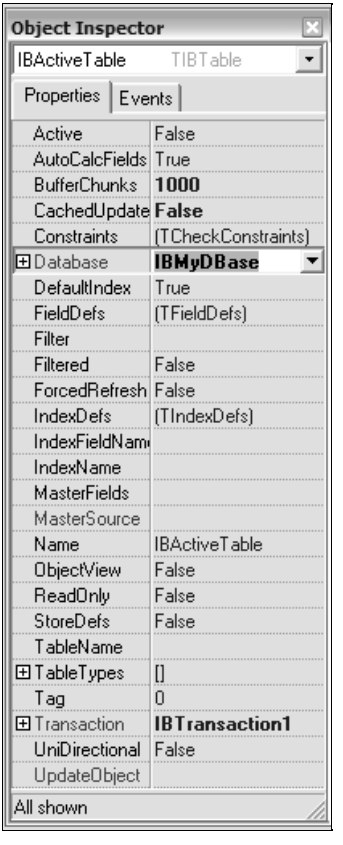


Рис. 7.44. Установка свойств для компонента `IBTable`

Для компонента `IBQuery` установим свойства, аналогичные компоненту `IBTable`:

- `Name` — присвоить значение `IBActiveQuery`;
- `Database` — присвоить значение `IBMyDBase` (псевдоним используемой базы данных);
- `IBTransaction` — присвоить значение `Transaction1`.

В клиентском приложении для вывода данных таблицы либо запроса в виде таблицы используется визуальный компонент DBGrid. Для связи визуального компонента DBGrid и компонентов IBTable и IBQuery необходимо использовать компонент DataSource. В разрабатываемом приложении размещаются на форме два компонента DataSource, один из которых связан с компонентом IBTable, а другой — IBQuery.

Установить для первого компонента DataSource следующие свойства:

- DataSet — присвоить значение IBActiveTable;
- Name — присвоить значение TableDataSource.

Установить для второго компонента DataSource следующие свойства:

- DataSet — присвоить значение IBActiveQuery;
- Name — присвоить значение QueryDataSource.

Для компонента DBGrid в свойстве DataSource необходимо установить значение TableDataSource (для вывода значений, содержащихся в таблицах базы данных).

Для выполнения базовых операций над записями в таблицах базы данных служит компонент DBNavigator. Для его связи с имеющейся базой данных следует присвоить свойству DataSource этого компонента значение TableDataSource.

Как отмечалось ранее, одной из функций разрабатываемой системы является возможность просмотра всех таблиц базы данных. Таким образом, пользователю должна быть предоставлена возможность выбора необходимой для просмотра таблицы. Для этого нужно добавить на форму компонент RadioGroup и присвоить ему имя TableRG (свойство Caption — Список таблиц либо List of Tables). Элементами данного компонента будут имена таблиц базы, определенной в свойствах компонента IBDatabase. Доступ к элементам можно получить, используя свойство Items данного компонента.

Здесь возможны два варианта.

- Указать с помощью инспектора объектов в качестве элементов объекта `RadioGroup` имена таблиц базы данных, используемой по умолчанию. Однако в этом случае данное приложение будет жестко привязано к конкретной базе данных и во время работы с системой, в нем нельзя работать с другой БД.
- Реализовать соответствующую процедуру, позволяющую обращаться к таблицам любой подключенной к приложению базы данных.

Для разрабатываемого приложения выберем второй вариант подключения таблиц активной базы данных. Итак, в обработчик события `OnShow` главной формы приложения (рис. 7.45) необходимо разместить следующий код:

```
IBMyDBase.GetTableNames(TableRG.Items);
// элементам объекта TableRG присваиваем имена таблиц
TableRg.ItemIndex:=0;
// активным устанавливаем первый элемент объекта TableRG
IBGrid.DataSource:=TableDataSource;
// в качестве источника данных для объекта IBGrid
// устанавливаем объект TableDataSource
IBActiveTable.TableName:=TableRg.Items[TableRG.ItemIndex];
// активной устанавливаем таблицу, соответствующую активному
// элементу объекта TableRG
IBActiveTable.Open();
// открываем активную таблицу
```

Добавим также на форму кнопку для подтверждения просмотра выбранной таблицы. Для этого выберем компонент `Button`, расположенный на вкладке **Standard**. Переименуем этот объект, присвоив его свойству `Name` новое значение `ShowTableBut` (`Caption` — `Показать таблицу` либо `Show Table`), и

в обработчике события `OnClick` этого компонента (двойной щелчок мыши по данной кнопке) запишем следующий код:

```
IBGrid.DataSource:=TableDataSource;
// в качестве источника данных для объекта IBGrid
// устанавливаем объект TableDataSource
IBActiveTable.Close();
// для избежания возможного конфликта закрываем активную
// таблицу
IBActiveTable.TableName:=TableRg.Items[TableRG.ItemIndex];
// активной устанавливаем таблицу, соответствующую выделенному
// элементу объекта TableRG
IBActiveTable.Open();
// открываем активную таблицу
```

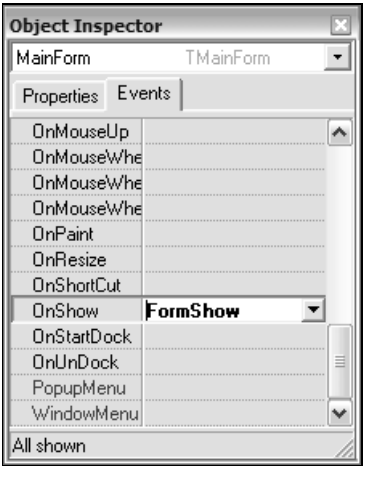


Рис. 7.45. Выбор события `OnShow` для главной формы приложения

Итак, с помощью данного приложения можно просматривать таблицы имеющейся базы данных music.gdb.

Далее реализуем в приложении возможность выполнения SQL-запросов.

Добавим на форму два компонента для работы с многострочным текстом:

- компонент `Memo` — расположен на вкладке **Standard** панели инструментов **Component Palette** и необходим для отображения текстового описания (в данном случае для описания SQL-запроса). Присвоим данному компоненту имя `QueryMemo` (для подписи здесь и далее используем компонент `GroupBox`, расположенный на вкладке **Standart** панели инструментов **Component Palette**, и изменяем соответственно свойство `Caption`; в данном случае `Поле имени` либо `Query NameField`).
- компонент `RichEdit` — находится на вкладке **Win32** панели инструментов **Component Palette** и служит для отображения самого SQL-запроса. Свойству `Name` данного компонента присвоим значение `REQuery` (`Поле текста запроса` либо `Query TextField`).

Кроме того, как указано ранее, система должна обладать возможностью сохранения запросов в виде текстовых файлов, а также их последующей загрузкой.

Для реализации таких функциональных возможностей необходимо наличие в приложении компонентов `OpenDialog` и `SaveDialog`, которые расположены в разделе **Dialogs** панели инструментов **Component Palette**.

Чтобы предоставить пользователю возможность выбора способа записи SQL-запроса, добавим на форму еще один компонент `RadioGroup`, которму присвоим имя `QueryRG` (Выполнить действие). В свойстве `Items` данного элемента добавим три записи (рис. 7.46):

- `Input your own` (Создать запрос самостоятельно);
- `Input using wizard` (Создать, используя мастер);
- `Load from file` (Загрузить из файла).

При выборе первого элемента (`Input your own`) пользователь сможет самостоятельно вводить SQL-запрос в соответствую-

щее поле (компонент REQuery). При выборе элемента Input using wizard построение запроса будет выполнено с помощью мастера построения запросов, реализация которого будет приведена далее. Последний элемент группы Load from file означает возможность загрузки пользователем SQL-запроса из файла.

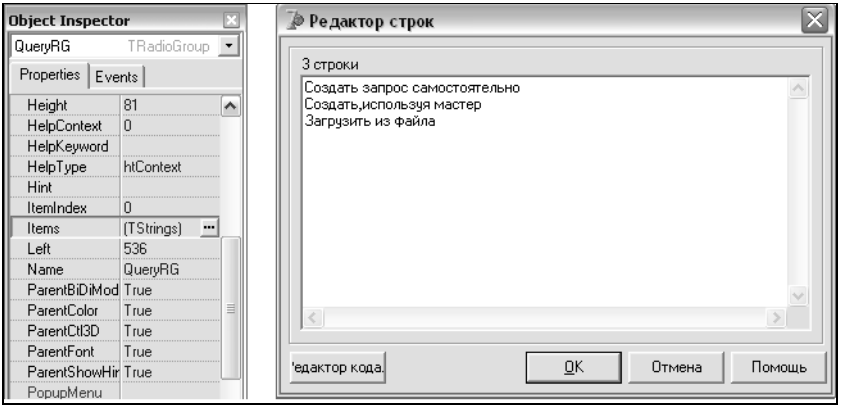


Рис. 7.46. Изменение свойства Items компонента QueryRG (RadioGroup)

Следует отметить, что для подтверждения выполнения SQL-запроса необходим компонент Button, который помещаем на форму и присваиваем его свойству Name значение MakeQueryBut (Выполнить запрос). Для подтверждения сохранения запроса в файл помещаем еще один объект Button с именем SaveQueryBut (Сохранить запрос).

Для выполнения запросов в обработчик события OnClick компонента MakeQueryBut введем следующий код:

```
procedure TMainForm.MakeQueryButClick(Sender: TObject);
var s:PAnsiChar;
    f:TextFile;
    str:string;
```

```
begin
  GB.Caption:='Вид запроса';
  case QueryRG.ItemIndex of
    0:begin
        IBActiveQuery.Close();
        s:=REQuery.Lines.GetText;
        IBActiveQuery.SQL.Text:=s;
        IBActiveQuery.Open();
        IBGrid.DataSource:=QueryDataSource;
      end;
    1:begin
        QueryCreatorForm.ShowModal;
      end;
    2:begin
        IBOpenDialog.Title:='Выберите файл...';
        IBOpenDialog.Filter:='*.qr';
        if IBOpenDialog.Execute then
          begin
            AssignFile(f,IBOpenDialog.FileName);
            Reset(f);
            readln(f,str);
            QueryMemo.Lines[0]:=str;
            while not eof(f) do
              begin
                readln(f,str);
                REQuery.Lines.Add(str);
              end;
            CloseFile(f);
          end;
        REQuery.Enabled:=true;
        REQuery.Color:=clWhite;
        QueryMemo.Enabled:=true;
        QueryMemo.Color:=clWhite;
```

```
        QueryRG.ItemIndex:=0;
        SBar.Panels[1].Text:='Запрос -
        '+IBOpenDialog.FileName;
      end;
  end;
end;
```

Для сохранения запроса, который создан пользователем на языке SQL, в обработчик события `OnClick` объекта `SaveQueryBut` добавим код:

```
procedure TMainForm.SaveQueryButClick(Sender: TObject);
var f:TextFile;
    str:string;
    s:TStrings;
    i:integer;
begin
  if IBSaveDialog.Execute then
    begin
      AssignFile(f,IBSaveDialog.FileName);
      Rewrite(f);
      writeln(f,QueryMemo.Lines.Text);
      i:=0;
      while (REQuery.Lines.Strings[i]<>'') do
        begin
          writeln(f,REQuery.Lines.Strings[i]);
          inc(i);
        end;
        CloseFile(f);
    end;
end;
```

Для того чтобы пользователь мог очистить поле ввода запроса, добавим один компонент `Button` с именем `ClearQueryBut` (Очистить запрос) и для реализации данной

функции создадим процедуру обработки события `OnClick` со следующим кодом:

```
procedure TMainForm.ClearQueryButClick(Sender: TObject);
begin
  REQuery.Clear;
  QueryMemo.Clear;
end;
```

Как правило, каждое приложение для удобства его использования имеет соответствующее меню. Для создания меню приложения добавим на форму компонент `MainMenu`, расположенный на вкладке **Standard** панели инструментов **Component Palette**.

При работе с линейкой меню будут предусмотрены следующие возможности:

- работа с базой данных:
  - подключение новой базы данных;
  - отключение базы данных;
  - выход их приложения;
- работа с запросами:
  - создать запрос самостоятельно;
  - создать, используя мастер;
  - загрузить из файла;
  - очистить поля запроса;
  - сохранить запрос;
- информация о приложении.

`MainMenu` — невизуальный компонент, основное свойство которого `Items` заполняется с использованием конструктора меню. Конструктор меню вызывается двойным щелчком мыши на компоненте `MainMenu` либо нажатием кнопки с многоточи-

ем рядом со свойством `Items` в окне инспектора объектов **(Object Inspector)** (рис. 7.47).

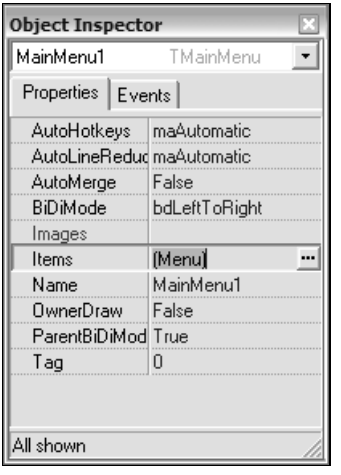


Рис. 7.47. Свойства компонента `MainMenu` в окне **Object Inspector**

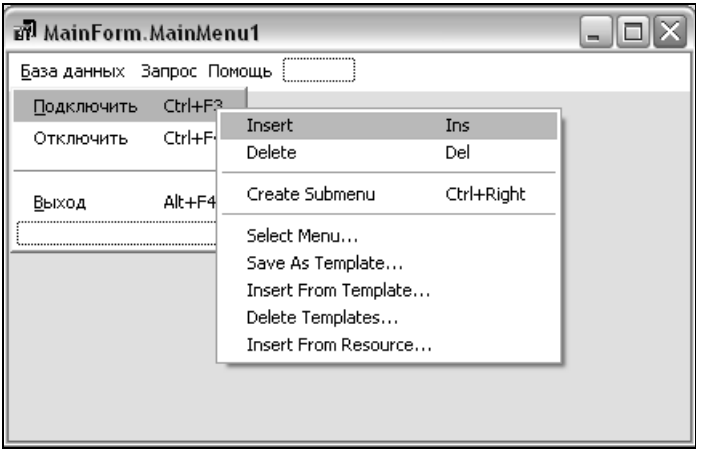


Рис. 7.48. Работа с конструктором меню

Новые разделы меню можно вводить, щелкнув правой кнопкой мыши и выбрав команду **Insert** (рис. 7.48).

После того как меню сконструировано, необходимо добавить код обработки событий (выбора соответствующих команд меню).

Раздел меню **Подключить** (либо Connect). Для раздела можно назначить "горячие" клавиши (сочетание клавиш), при нажатии которых будут выполняться такие же действия, как и при выборе данного пункта меню (рис. 7.49).

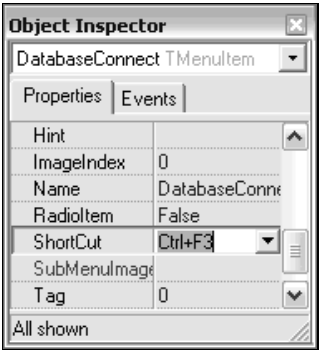


Рис. 7.49. Назначение сочетаний клавиш для быстрого выполнения пункта меню

Для обработки события `OnClick` данного раздела необходимо ввести соответствующий код:

```
procedure TMainForm.DatabaseConnectClick(Sender: TObject);
begin
 IBOpenDialog.Title:='Âûáåðèòå ôàéë...';
 IBOpenDialog.Filter:='*.gdb';
  if IBOpenDialog.Execute then
    begin
      IBMyDBase.Close();
      IBMyDBase.DatabaseName:=IBOpenDialog.FileName;
```

```
      GB.Enabled:=True;
        TableRG.Enabled:=True;
        IBGrid.Enabled:=True;
        IBGrid.Color:=clWhite;
        ShowTableBut.Enabled:=True;
      CreateQueryGB.Enabled:=True;
        ClearQueryBut.Enabled:=True;
        MakeQueryBut.Enabled:=True;
        QueryRG.Enabled:=True;
        REQuery.Color:=clWhite;
        QueryMemo.Enabled:=true;
        QueryMemo.Color:=clWhite;
      FormShow(Sender);
    end;
end;
```

Раздел меню **Отключить** (либо Disconnect). Для обработки события `OnClick` необходимо ввести следующий код:

```
procedure TMainForm.DatabaseDisconnectClick(Sender: TObject);
begin
  IBMyDbase.Close;
  GB.Enabled:=False;
    TableRG.Enabled:=false;
    IBGrid.Enabled:=false;
    IBGrid.Color:=clMenu;
    ShowTableBut.Enabled:=False;
  CreateQueryGB.Enabled:=False;
    ClearQueryBut.Enabled:=False;
    MakeQueryBut.Enabled:=False;
    QueryRG.Enabled:=False;
    REQuery.Color:=clMenu;
    QueryMemo.Color:=clMenu;
end;
```

Раздел меню **Выход** (либо Exit). Для обработки события `OnClick` добавляется код:

```
ExitBut.Click;
```

Раздел меню **Создать запрос** (Make query) с подразделами:

- **Создать запрос самостоятельно** (Input your own);
- **Создать, используя мастер** (Input using wizard);
- **Загрузить из файла** (Load from file).

Для обработки события `OnClick` подраздела **Input your own** необходимо ввести код:

```
QueryRG.ItemIndex:=0;
MakeQueryBut.Click();
```

Для пунктов **Input using wizard** и **Load from file** выполняем аналогичные действия.

Для разделов **Очистить поле запроса** (Clear query fild) и **Сохранить как** (Save as) вызываем события нажатия соответствующих клавиш (описаны ранее).

Создание раздела меню **Помощь** (Help) будет описано далее.

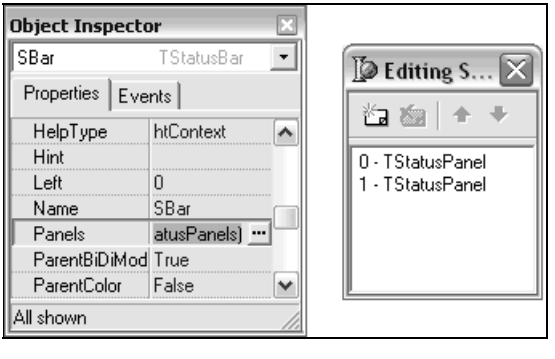


Рис. 7.50. Добавление разделов в свойство `Panels` компонента `StatusBar`

Для вывода информации о текущей загруженной базе и загруженном запросе добавим на форму компонент `StatusBar`, расположенный на вкладке **Win32** панели инструментов **Component Palette**. Для свойства `Panels` данного компонента добавим два раздела: `0 – TStatusPanel` и `1 – TStatusPanel` (рис. 7.50).

Таким образом, главная форма имеет соответствующие компоненты и представлена на рис. 7.51.

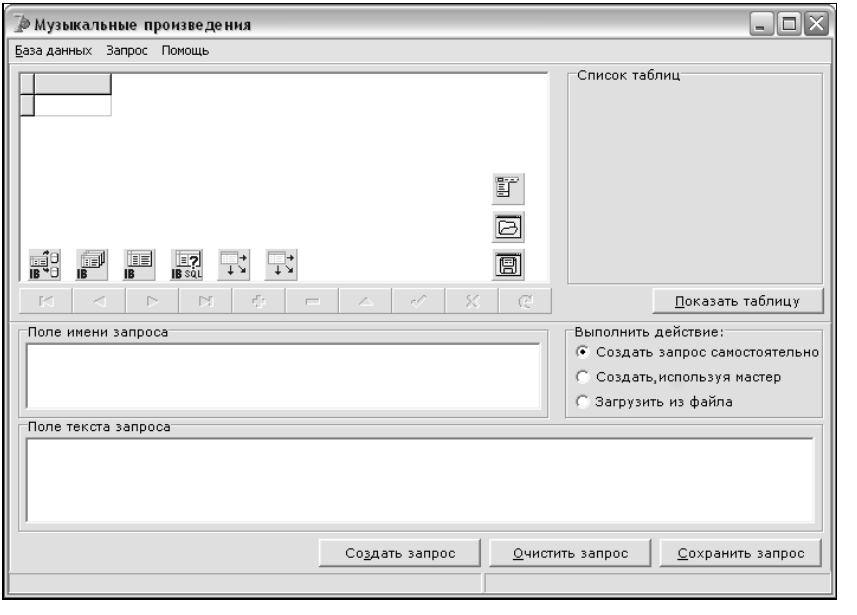


Рис. 7.51. Главная форма приложения на стадии разработки

Для обработки события, связанного с компонентами `QueryRG` (Выполнить действие) и `REQuery` (Поле текста запроса), введем следующий код:

```
procedure TMainForm.QueryRGClick(Sender: TObject);
begin
  case QueryRG.ItemIndex of
```

```
    0:begin
        MakeQueryBut.Caption:='&Выполнить запрос';
        REQuery.Enabled:=true;
        REQuery.Color:=clWhite;
        QueryMemo.Enabled:=true;
        QueryMemo.Color:=clWhite;
      end;
    1:begin
        MakeQueryBut.Caption:='&Выполнить запрос';
        REQuery.Enabled:=false;
        REQuery.Color:=clBtnFace;
        QueryMemo.Enabled:=false;
        QueryMemo.Color:=clBtnFace;
      end;
    2:begin
        MakeQueryBut.Caption:='Загрузить запрос';
        REQuery.Enabled:=false;
        REQuery.Color:=clBtnFace;
        QueryMemo.Enabled:=false;
        QueryMemo.Color:=clBtnFace;
      end;
  end;
end;
```

В случае, когда пользователь выберет элемент группы **Создать, используя мастер** компонента `QueryRG`, построение запроса будет выполнено с помощью мастера построения запросов.

Мастер построения запросов

Для реализации мастера построения запросов в проект необходимо добавить новую форму.

Сформулируем основные требования, предъявляемые к мастеру запросов:

- возможность выбора необходимой таблицы;
- возможность выбора необходимых полей таблицы;
- возможность выбора ключевых слов:
  - `select`, `insert`, `delete`, `update`, `from`, `where`, `having by`, `order by`, `all`, `distinct`, `into`, `values`, `set`, `AND`, `NOT`, `OR`.

Необходимые компоненты для реализации мастера запросов:

- `IBTable` (вкладка **Interbase**);
- `RichEdit` (вкладка **Win32**);
- `ListBox` (вкладка **Standard**);
- `RadioGroup` (вкладка **Standard**);
- `StringGrid` (вкладка **Additional**);
- `Button` (вкладка **Standard**)
- `GroupBox` (вкладка **Standard**).

Новая форма добавляется в проект с использованием команды меню **File** | **New** | **Form** (рис. 7.52).

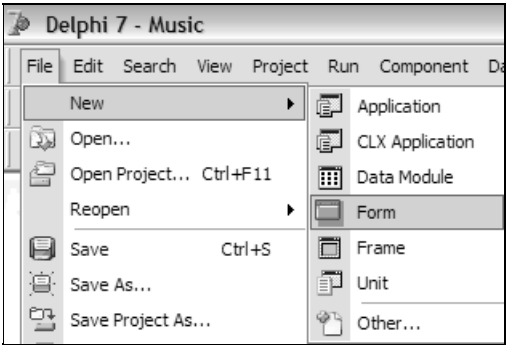


Рис. 7.52. Добавление новой формы в проект

Сохраним созданный файл под именем `QueryCreator`.

На новую форму поместим следующие компоненты:

- компонент `RichEdit` — для отображения запроса, вводимого с помощью мастера построения или с клавиатуры. Присвоим ему имя `QueryRE`;
- компонент `IBTable` — для связи с базой данных. Определим его основные свойства (рис. 7.53):
  - свойство `Database` — `MainForm.IBMyDBase`;

Рис. 7.53. Определение свойств компонента `IBTable`

- свойство MasterSourse — MainForm.TableDataSource;
- свойство Name — IBChoozenTable;
- свойство Transaction — MainForm.IBTransaction1;

❑ два компонента ListBox:

- для отображения таблиц базы данных используется компонент с именем TableLB (рис. 7.54);

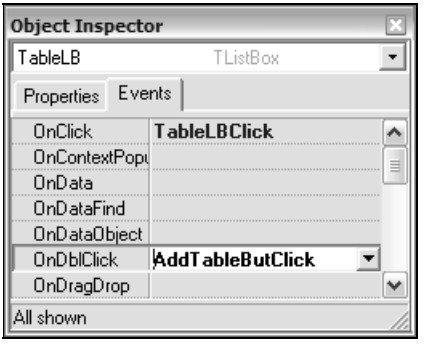


Рис. 7.54. Определение событий для компонента ListBox с именем TableLB

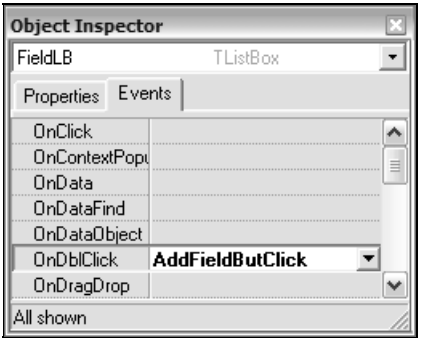


Рис. 7.55. Определение событий для компонента ListBox с именем FieldLB

- для отображения названия полей соответствующих таблиц базы данных используется компонент с именем FieldLB (рис. 7.55);

- компонент RadioGroup (свойство Name — Query TypeRG) — для отображения типа вводимых запросов;
- компонент StringGrid (свойство Name — KeyWordSG) — для отображения списка ключевых слов, которые можно включать в запрос;
- четыре компонента Button:
  - для добавления таблицы (Name — AddTableBut) — Добавить таблицу (Add table);
  - для добавления полей таблицы (Name — AddFieldBut) — Добавить поле (Add field);
  - для очистки поля ввода запроса (Name — ClearBut) — Очистить запрос (Clear query);
  - для выполнения запроса (Name — MakeQueryBut) — Выполнить запрос (Make query);
- три компонента GroupBox для соответствующих подписей разделов мастера (изменяем соответственно свойство Caption): Выберите таблицу (Select table), Выберите поле (Select field) и Запрос (Query).

Таким образом, форма для мастера построения запросов может выглядеть, например, так как показано на рис. 7.56.

Теперь добавим необходимый код для обработки событий, связанных с мастером построения запросов.

Для события OnShow формы мастера запросов:

```
procedure TQueryCreatorForm.FormShow(Sender: TObject);
var i:integer;
begin
  MainForm.IBMyDBase.GetTableNames(TableLB.Items);
```

```
  TableLB.ItemIndex:=0;
  IBChoozenTable.TableName:=TableLB.Items[TableLB.ItemIndex];
  IBChoozenTable.GetFieldNames(FieldLB.Items);

  QueryTypeRG.ItemIndex:=0;
  KeyWordsSG.Cells[0,0]:='Ключевые слова';
  for i:=1 to 22 do
    KeyWordsSG.Cells[0,i]:=words[i];
end;
```

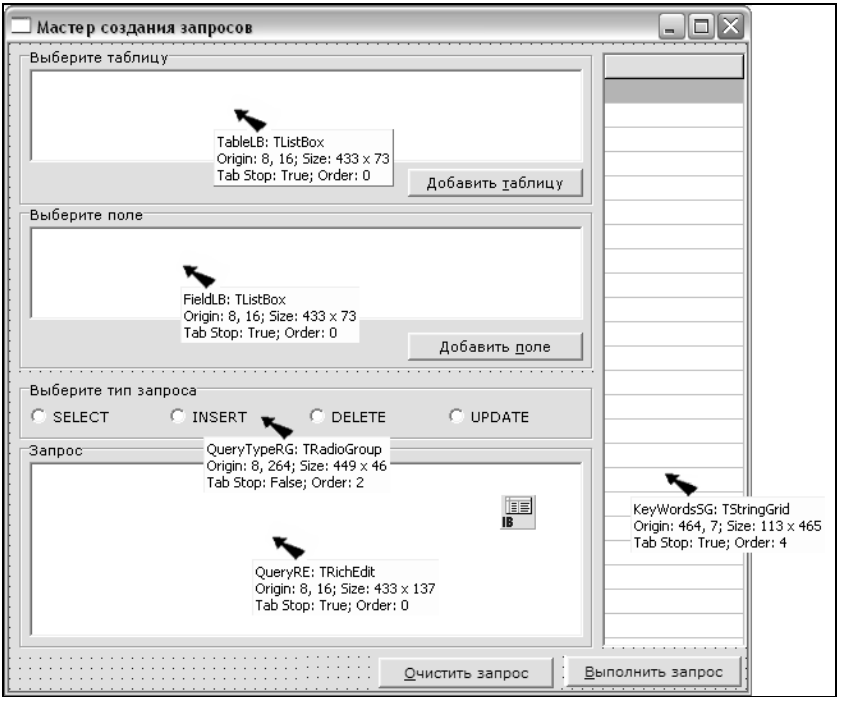


Рис. 7.56. Форма для построения мастера запросов на стадии разработки

Для ключевых слов, которые могут использоваться мастером запросов, определяем соответствующий массив-переменную:

```
var
  QueryCreatorForm: TQueryCreatorForm;
      words: array [1..22] of string =
       ('select','insert','delete','update',
        'from','where','having by','order by',
         'all','distinct','into','values',
         'set','AND','NOT','OR',
         '+','-','=','<>',
         '*','/');
```

Для события `OnClick` по компоненту `TableLB` (выбор таблицы):

```
 procedure TQueryCreatorForm.TableLBClick(Sender: TObject);
begin
  IBChoozenTable.TableName:=TableLB.Items[TableLB.ItemIndex];
  IBChoozenTable.GetFieldNames(FieldLB.Items);
  FieldLB.ItemIndex:=0;
end;
```

Для события `OnClick` кнопки `AddTableBut`:

```
procedure TQueryCreatorForm.AddTableButClick(Sender:
TObject);
begin
  if HelpCheckBox.Checked=true then
   case (QueryTypeRG.ItemIndex) of
    0:begin
        if QueryRE.FindText('select',0,1000,[stWholeWord])
        <>-1 then
        if QueryRE.FindText('from',0,1000,[stWholeWord])=-1
then
          QueryRE.Lines.Append('    from '+
          TableLB.Items[TableLB.ItemIndex])
```

```
        else QueryRE.Lines.Append('          , '+
                              TableLB.Items[TableLB.ItemIndex])
      end;
    1:begin
        QueryRE.Clear;
        QueryRE.Lines.Append('insert into '+
                      TableLB.Items[TableLB.ItemIndex]+ ' ');
      end;
    2:begin
        QueryRE.Clear;
        QueryRE.Lines.Append('delete from '+
                      TableLB.Items[TableLB.ItemIndex]+ ' ');
      end;
    3:begin
        QueryRE.Clear;
        QueryRE.Lines.Append('update '+
        TableLB.Items[TableLB.ItemIndex]+ ' set ');
      end;
  end
  else QueryRE.Lines.Append(
  TableLB.Items[TableLB.ItemIndex]);
end;
```

Для события `OnClick` кнопки `AddFieldBut`:

```
procedure TQueryCreatorForm.AddFieldButClick(Sender: TOb-
ject);
var ch:char;
begin
  if HelpCheckBox.Checked=true then
    case (QueryTypeRG.ItemIndex) of
      0:begin
        if QueryRE.FindText('select',0,1000,
        [stWholeWord])=-1 then
```

```
            begin
              QueryRE.Clear;
              QueryRE.Lines.Append('select ' +

FieldLB.Items[FieldLB.ItemIndex]);
            end
          else
          if QueryRE.FindText('from',0,1000,[stWholeWord])=-1
then
            begin
              QueryRE.Lines.APPEND('       , ' +

FieldLB.Items[FieldLB.ItemIndex]);
            end
          else
          if QueryRE.FindText('where',0,1000,
          [stWholeWord])=-1 then
            begin
             QueryRE.Lines.APPEND('        where ' +
                          FieldLB.Items[FieldLB.ItemIndex])
            end
          else
          if QueryRE.FindText('group by',0,1000,
                            [stWholeWord])=-1 then
            begin
         QueryRE.Lines.APPEND('        group by ' +

FieldLB.Items[FieldLB.ItemIndex]);
            end
          else
           if QueryRE.FindText('having',0,1000,
                             [stWholeWord])=-1 then
            begin
           QueryRE.Lines.APPEND('        having ' +
           FieldLB.Items[FieldLB.ItemIndex]);
```

```
            end
          else
          if QueryRE.FindText('order by',0,1000,
                              [stWholeWord])=-1 then
            begin
         QueryRE.Lines.APPEND('        order by ' +

FieldLB.Items[FieldLB.ItemIndex]);
            end
        end;
      1:begin
          if QueryRE.FindText('insert into',
                               0,1000,[stWholeWord])<>-1 then
          QueryRE.Lines.Append(FieldLB.Items
          [FieldLB.ItemIndex]);
        end;
      2:begin
          if QueryRE.FindText('delete from',
                               0,1000,[stWholeWord])<>-1 then
            QueryRE.Lines.Append('         where '+

FieldLB.Items[FieldLB.ItemIndex]);
        end;
      3:begin
          if QueryRE.FindText('update',0,1000,
          [stWholeWord])<>-1 then
            QueryRE.Lines.Append('         '+
                          FieldLB.Items[FieldLB.ItemIndex]);
        end;
    end
  else QueryRE.Lines.Append(FieldLB.Items
 [FieldLB.ItemIndex]);
end;
```

Для события `OnClick` кнопки `MakeQueryBut`:

```
procedure TQueryCreatorForm.MakeQueryButClick(Sender:
TObject);
var s:string;
begin
  MainForm.REQuery.Lines:=QueryRE.Lines;
  MainForm.QueryRG.ItemIndex:=0;
  QueryCreatorForm.Close;
end;
```

Для события `OnSelectCell` компонента `KeyWordsSG`:

```
procedure TQueryCreatorForm.KeyWordsSGSelectCell(Sender:
TObject; ACol,
 ARow: Integer; var CanSelect: Boolean);
begin
 QueryRE.Lines.Append(KeyWordsSG.Cells[ACol,Arow]);
end;
```

```
Для события OnClick кнопки ClearBut:
procedure TQueryCreatorForm.ClearButClick(Sender: TObject);
begin
 QueryRE.Clear;
end;
```

Таким образом, создание мастера построения запросов завершено.

По аналогии можно добавить в проект (используя построение новых форм и связывая их с определенными событиями) информацию о программе и т. д.

После того как основные действия по созданию проекта завершены, проект необходимо скомпилировать, используя со-

ответствующую команду категории меню **Project** (либо сочетание клавиш <Ctrl>+<F9>).

Листинг клиентского приложения

Листинг клиентского приложения, предназначенного для работы с базой данных "Музыкальные произведения", приводится далее.

Главная форма

Внешний вид главной формы показан на рис. 7.57.

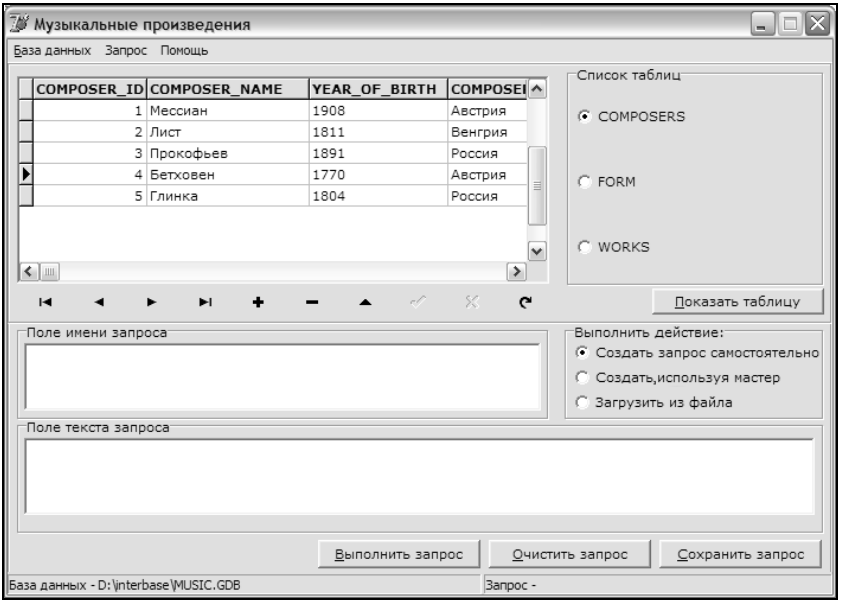


Рис. 7.57. Окно клиентского приложения "Музыкальные произведения"

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,
  Dialogs, IBSQL, DB, IBDatabase, IBQuery, IBCustomDataSet,
  IBTable, Grids,
  DBGrids, StdCtrls, DBCtrls, ExtCtrls, ComCtrls, OleServer,
  Word2000,
  Menus, ToolWin, ImgList,IB;

type
  TMainForm = class(TForm)
    IBMyDBase: TIBDatabase;
    IBActiveTable: TIBTable;
    TableDataSource: TDataSource;
    IBActiveQuery: TIBQuery;
    QueryDataSource: TDataSource;
    IBTransaction1: TIBTransaction;
    CreateQueryGB: TGroupBox;
    MakeQueryBut: TButton;
    GB: TGroupBox;
    IBNavigator: TDBNavigator;
    IBGrid: TDBGrid;
    TableRG: TRadioGroup;
    ShowTableBut: TButton;
    ClearQueryBut: TButton;
    MainMenu1: TMainMenu;
    MMDatabase: TMenuItem;
    DatabaseConnect: TMenuItem;
    DatabaseDisconnect: TMenuItem;
    N1: TMenuItem;
    DatabaseExit: TMenuItem;
    IBOpenDialog: TOpenDialog;
```

```
IBSaveDialog: TSaveDialog;
QueryRG: TRadioGroup;
MMQuery: TMenuItem;
MakeQuery: TMenuItem;
Inputyourown1: TMenuItem;
Inputusingwizard1: TMenuItem;
LoadFromFile: TMenuItem;
ClearQueryFild: TMenuItem;
N3: TMenuItem;
QuerySaveAs: TMenuItem;
MMHelp: TMenuItem;
HelpAbout: TMenuItem;
SBar: TStatusBar;
SaveQueryBut: TButton;
NameFieldGB: TGroupBox;
TextFieldGB: TGroupBox;
QueryMemo: TMemo;
REQuery: TRichEdit;
procedure ShowTableButClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure MakeQueryButClick(Sender: TObject);
procedure REQueryKeyPress(Sender: TObject; var Key:
Char);


procedure ExitButClick(Sender: TObject);
procedure ClearQueryButClick(Sender: TObject);
procedure DatabaseExitClick(Sender: TObject);
procedure DatabaseConnectClick(Sender: TObject);
procedure DatabaseDisconnectClick(Sender: TObject);
procedure Inputyourown1Click(Sender: TObject);
procedure Inputusingwizard1Click(Sender: TObject);
procedure LoadFromFileClick(Sender: TObject);
procedure ClearQueryFildClick(Sender: TObject);
procedure QueryRGClick(Sender: TObject);
```

```
    procedure HelpAboutClick(Sender: TObject);
    procedure SaveQueryButClick(Sender: TObject);
    procedure QuerySaveAsClick(Sender: TObject);


  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
implementation

uses QueryCreator, About1;

{$R *.dfm}

procedure TMainForm.ShowTableButClick(Sender: TObject);
begin
  IBGrid.DataSource:=TableDataSource;
  IBActiveTable.Close();
  IBActiveTable.TableName:=TableRg.Items[TableRG.ItemIndex];
  IBActiveTable.Open();
end;

function ONMD(Sender: TStringGrid;Button: TMouseButton;
Shift: TShiftState;X,Y:integer):TMouseEvent;
{$IFNDEF WIN32}
var pt : TPoint;
{$ENDIF}
begin
if ssCtrl in Shift then begin
  ReleaseCapture;
```

```
  SendMessage(Sender.Handle, WM_SYSCOMMAND, 61458, 0);
  {$IFNDEF WIN32}
  GetCursorPos(pt);
  SendMessage(Sender.Handle,WM_LBUTTONUP,MK_CONTROL,
  Longint(pt));
  {$ENDIF}
end;
end;

procedure TMainForm.FormShow(Sender: TObject);
begin
  IBMyDBase.GetTableNames(TableRG.Items);
  TableRg.ItemIndex:=0;
  IBGrid.DataSource:=TableDataSource;
  IBActiveTable.TableName:=TableRg.Items[TableRG.ItemIndex];
  SBar.Panels[0].Text:='База данных - '+
  IBMyDBase.DatabaseName;
  SBar.Panels[1].Text:='Запрос - ';
  IBActiveTable.Open();
end;



procedure TMainForm.MakeQueryButClick(Sender: TObject);
var s:PAnsiChar;
    f:TextFile;
    str:string;
begin
  GB.Caption:='Вид запроса';
  case QueryRG.ItemIndex of
    0:begin
        IBActiveQuery.Close();
        s:=REQuery.Lines.GetText;
        IBActiveQuery.SQL.Text:=s;
        IBActiveQuery.Open();
        IBGrid.DataSource:=QueryDataSource;
      end;
```

```
    1:begin
        QueryCreatorForm.ShowModal;
      end;
    2:begin
        IBOpenDialog.Title:='Выберите файл...';
        IBOpenDialog.Filter:='*.qr';
        if IBOpenDialog.Execute then
          begin
            AssignFile(f,IBOpenDialog.FileName);
            Reset(f);
            readln(f,str);
            QueryMemo.Lines[0]:=str;
            while not eof(f) do
              begin
                readln(f,str);
                REQuery.Lines.Add(str);
              end;
            CloseFile(f);
          end;
        REQuery.Enabled:=true;
        REQuery.Color:=clWhite;
        QueryMemo.Enabled:=true;
        QueryMemo.Color:=clWhite;
        QueryRG.ItemIndex:=0;
        SBar.Panels[1].Text:='Запрос -
        '+IBOpenDialog.FileName;
      end;
  end;
end;

procedure TMainForm.REQueryKeyPress(Sender: TObject; var Key:
Char);
var pos:integer;
begin
{   pos:=RE.FindText('select',0,10,[stWholeWord]);
```

```
    RE.DefAttributes.Color:=clblack;
  if pos > -1 then
    begin
      RE.SelAttributes.Color:=clred;
    end;}
end;



procedure TMainForm.ExitButClick(Sender: TObject);
begin
  if MessageDlg('Закрыть
  приложение?',mtConfirmation,[mbYes,mbNo],0)=mrYes
  then MainForm.Close  ;
end;

procedure TMainForm.ClearQueryButClick(Sender: TObject);
begin
  REQuery.Clear;
  QueryMemo.Clear;
end;

procedure TMainForm.DatabaseExitClick(Sender: TObject);
begin
  {ExitBut.Click;}
end;

procedure TMainForm.DatabaseConnectClick(Sender: TObject);
begin
 IBOpenDialog.Title:='Выберите файл...';
 IBOpenDialog.Filter:='*.qdb';
  if IBOpenDialog.Execute then
    begin
      IBMyDBase.Close();
      IBMyDBase.DatabaseName:=IBOpenDialog.FileName;
      GB.Enabled:=True;
```

```
        TableRG.Enabled:=True;
        IBGrid.Enabled:=True;
        IBGrid.Color:=clWhite;
        ShowTableBut.Enabled:=True;
      CreateQueryGB.Enabled:=True;
        ClearQueryBut.Enabled:=True;
        MakeQueryBut.Enabled:=True;
        QueryRG.Enabled:=True;
        REQuery.Color:=clWhite;
        QueryMemo.Enabled:=true;
        QueryMemo.Color:=clWhite;
      FormShow(Sender);
    end;
end;

procedure TMainForm.DatabaseDisconnectClick(Sender: TObject);
begin
  IBMyDbase.Close;
  GB.Enabled:=False;
    TableRG.Enabled:=false;
    IBGrid.Enabled:=false;
    IBGrid.Color:=clMenu;
    ShowTableBut.Enabled:=False;
  CreateQueryGB.Enabled:=False;
    ClearQueryBut.Enabled:=False;
    MakeQueryBut.Enabled:=False;
    QueryRG.Enabled:=False;
    REQuery.Color:=clMenu;
    QueryMemo.Color:=clMenu;
end;

procedure TMainForm.Inputyourown1Click(Sender: TObject);
begin
  QueryRG.ItemIndex:=0;
  MakeQueryBut.Click();
end;
```

```
procedure TMainForm.Inputusingwizard1Click(Sender: TObject);
begin
  QueryRG.ItemIndex:=1;
  MakeQueryBut.Click();
end;

procedure TMainForm.LoadFromFileClick(Sender: TObject);
begin
  QueryRG.ItemIndex:=2;
  MakeQueryBut.Click();
end;

procedure TMainForm.ClearQueryFildClick(Sender: TObject);
begin
  ClearQueryBut.Click();
end;

procedure TMainForm.QueryRGClick(Sender: TObject);
begin
  case QueryRG.ItemIndex of
    0:begin
        MakeQueryBut.Caption:='Со&здать запрос';
        REQuery.Enabled:=true;
        REQuery.Color:=clWhite;
        QueryMemo.Enabled:=true;
        QueryMemo.Color:=clWhite;
      end;
    1:begin
        MakeQueryBut.Caption:='Со&здать запрос';
        REQuery.Enabled:=false;
        REQuery.Color:=clBtnFace;
        QueryMemo.Enabled:=false;
        QueryMemo.Color:=clBtnFace;
      end;
```

```
    2:begin
        MakeQueryBut.Caption:='Загрузить запрос';
        REQuery.Enabled:=false;
        REQuery.Color:=clBtnFace;
        QueryMemo.Enabled:=false;
        QueryMemo.Color:=clBtnFace;
      end;
  end;
end;

procedure TMainForm.HelpAboutClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;

procedure TMainForm.SaveQueryButClick(Sender: TObject);
var f:TextFile;
    str:string;
    s:TStrings;
    i:integer;
begin
  if IBSaveDialog.Execute then
    begin
      AssignFile(f,IBSaveDialog.FileName);
      Rewrite(f);
      writeln(f,QueryMemo.Lines.Text);
      i:=0;
      while (REQuery.Lines.Strings[i]<>'') do
        begin
          writeln(f,REQuery.Lines.Strings[i]);
          inc(i);
        end;
        CloseFile(f);
    end;
end;
```

```
procedure TMainForm.QuerySaveAsClick(Sender: TObject);
begin
  SaveQueryBut.Click;
end;

end.
```

Мастер построения запросов

Внешний вид мастера построения запросов показан на рис. 7.58.

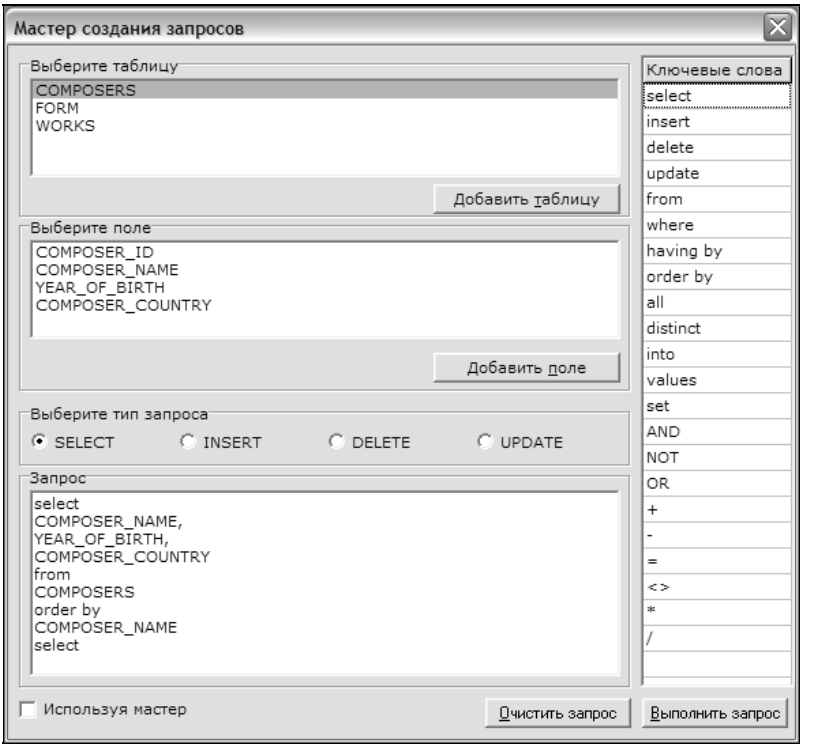


Рис. 7.58. Окно мастера построения запросов клиентского приложения "Музыкальные произведения"

```
unit QueryCreator;


interface


uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, ExtCtrls, DB, IBCustomDataSet,
  IBTable,Qt,
  Grids;
type
  TQueryCreatorForm = class(TForm)
    TableGB: TGroupBox;
    TableLB: TListBox;
    AddTableBut: TButton;
    FieldGB: TGroupBox;
    FieldLB: TListBox;
    AddFieldBut: TButton;
    QueryTypeRG: TRadioGroup;
    IBChoozenTable: TIBTable;
    GroupBox1: TGroupBox;
    QueryRE: TRichEdit;
    KeyWordsSG: TStringGrid;
    HelpCheckBox: TCheckBox;
    MakeQueryBut: TButton;
    ClearBut: TButton;
    procedure FormShow(Sender: TObject);
    procedure TableLBClick(Sender: TObject);
    procedure AddFieldButClick(Sender: TObject);
    procedure AddTableButClick(Sender: TObject);
    procedure MakeQueryButClick(Sender: TObject);
    procedure KeyWordsSGSelectCell(Sender: TObject; ACol,
    ARow: Integer;
      var CanSelect: Boolean);
```

```
    procedure ClearButClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  QueryCreatorForm: TQueryCreatorForm;
      words:array[1..22]of
string=('select','insert','delete','update',
                                  'from','where','having
by','order by',

'all','distinct','into','values',
                                  'set','AND','NOT','OR',
                                  '+','-','=','<>',
                                  '*','/');
implementation

uses Main;

{$R *.dfm}



procedure TQueryCreatorForm.FormShow(Sender: TObject);
var i:integer;
begin
  MainForm.IBMyDBase.GetTableNames(TableLB.Items);
  TableLB.ItemIndex:=0;
  IBChoozenTable.TableName:=TableLB.Items[TableLB.ItemIndex];
  IBChoozenTable.GetFieldNames(FieldLB.Items);

  QueryTypeRG.ItemIndex:=0;
  KeyWordsSG.Cells[0,0]:='Ключевые слова';
```

```
  for i:=1 to 22 do
    KeyWordsSG.Cells[0,i]:=words[i];
end;

procedure TQueryCreatorForm.TableLBClick(Sender: TObject);
begin
  IBChoozenTable.TableName:=TableLB.Items[TableLB.ItemIndex];
  IBChoozenTable.GetFieldNames(FieldLB.Items);
  FieldLB.ItemIndex:=0;
end;
procedure TQueryCreatorForm.AddFieldButClick(Sender:
TObject);
var ch:char;
begin
  if HelpCheckBox.Checked=true then
    case (QueryTypeRG.ItemIndex) of
      0:begin
          if QueryRE.FindText('select',0,1000,
          [stWholeWord])=-1 then
            begin
              QueryRE.Clear;
              QueryRE.Lines.Append('select ' +
              FieldLB.Items[FieldLB.ItemIndex]);
            end
          else
          if QueryRE.FindText('from',0,1000,
          [stWholeWord])=-1 then
            begin
              QueryRE.Lines.APPEND('       , ' +
              FieldLB.Items[FieldLB.ItemIndex]);
            end
          else
          if QueryRE.FindText('where',0,1000,
          [stWholeWord])=-1 then
```

```
        begin
          QueryRE.Lines.APPEND('        where ' +
          FieldLB.Items[FieldLB.ItemIndex])
        end
      else
      if QueryRE.FindText('group by',0,1000,
      [stWholeWord])=-1 then
        begin
          QueryRE.Lines.APPEND('        group by ' +
          FieldLB.Items[FieldLB.ItemIndex]);
        end
     else
      if QueryRE.FindText('having',0,1000,
      [stWholeWord])=-1 then
        begin
          QueryRE.Lines.APPEND('        having ' +
          FieldLB.Items[FieldLB.ItemIndex]);
        end
      else
      if QueryRE.FindText('order by',0,1000,
      [stWholeWord])=-1 then
        begin
          QueryRE.Lines.APPEND('        order by ' +
          FieldLB.Items[FieldLB.ItemIndex]);
        end
    end;
  1:begin
      if QueryRE.FindText('insert
      into',0,1000,[stWholeWord])<>-1 then
      QueryRE.Lines.Append(FieldLB.Items
      [FieldLB.ItemIndex]);
    end;
  2:begin
      if QueryRE.FindText('delete from',0,1000,
      [stWholeWord])<>-1 then
```

```
            QueryRE.Lines.Append('         where
            '+FieldLB.Items[FieldLB.ItemIndex]);
        end;
      3:begin
          if QueryRE.FindText('update',0,1000,
          [stWholeWord])<>-1 then
            QueryRE.Lines.Append('
            '+FieldLB.Items[FieldLB.ItemIndex]);
        end;
    end
  else QueryRE.Lines.Append(FieldLB.Items
  [FieldLB.ItemIndex]);
end;


procedure TQueryCreatorForm.AddTableButClick(Sender:
TObject);
begin
  if HelpCheckBox.Checked=true then
   case (QueryTypeRG.ItemIndex) of
    0:begin
        if QueryRE.FindText('select',0,1000,
        [stWholeWord])<>-1 then
        if QueryRE.FindText('from',0,1000,
        [stWholeWord])=-1 then
          QueryRE.Lines.Append('    from '+
          TableLB.Items[TableLB.ItemIndex])
        else QueryRE.Lines.Append('          , '+
        TableLB.Items[TableLB.ItemIndex])
      end;
    1:begin
        QueryRE.Clear;
        QueryRE.Lines.Append('insert into '+
        TableLB.Items[TableLB.ItemIndex]+ ' ');
      end;
    2:begin
        QueryRE.Clear;
```

```
        QueryRE.Lines.Append('delete from '+
        TableLB.Items[TableLB.ItemIndex]+ ' ');
      end;
    3:begin
        QueryRE.Clear;
        QueryRE.Lines.Append('update '+
        TableLB.Items[TableLB.ItemIndex]+ ' set ');
      end;
  end
  else QueryRE.Lines.Append(TableLB.Items
  [TableLB.ItemIndex]);
end;

procedure TQueryCreatorForm.MakeQueryButClick(Sender:
TObject);
var s:string;
begin
  MainForm.REQuery.Lines:=QueryRE.Lines;
  MainForm.QueryRG.ItemIndex:=0;
  QueryCreatorForm.Close;
end;
procedure TQueryCreatorForm.KeyWordsSGSelectCell(Sender:
TObject; ACol,
  ARow: Integer; var CanSelect: Boolean);
begin
  QueryRE.Lines.Append(KeyWordsSG.Cells[ACol,Arow]);
end;

procedure TQueryCreatorForm.ClearButClick(Sender: TObject);
begin
  QueryRE.Clear;
end;

end.
```

Информация о программе

Окно с информацией о созданной программе представлено на рис. 7.59.

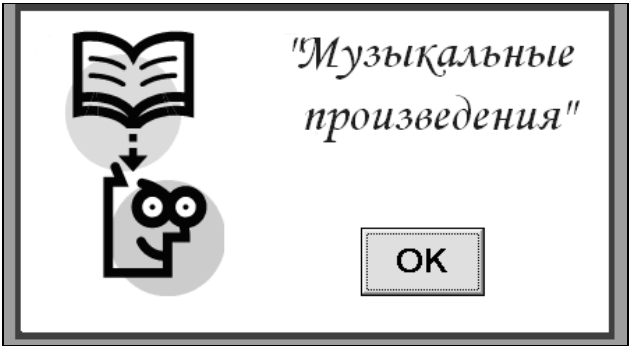


Рис. 7.59. Окно, выводящее информацию о программе "Музыкальные произведения"

```
unit About1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Buttons;

type
  TAboutForm = class(TForm)
    Image1: TImage;
    Button1: TButton;
    procedure Image1Click(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  AboutForm: TAboutForm;

implementation

{$R *.dfm}

procedure TAboutForm.Image1Click(Sender: TObject);
begin
  Close;
end;

end.
```

Задания

1. Какие основные подходы используются при проектировании под архитектуру "клиент-сервер"?
2. В чем заключаются различия между правилами для данных, правилами для процессов и правилами для интерфейса?
3. Что следует размещать при двухзвенной архитектуре на сторонах сервера и клиента?
4. Привести примеры использования двухзвенной архитектуры "клиент-сервер".
5. Дать характеристику трехзвенной архитектуры "клиент-сервер".
6. В чем главное отличие двухзвенной архитектуры "клиент-сервер" от трехзвенной.

7. Какие основные компоненты используются при создании клиентских приложений в архитектуре "клиент-сервер"?

8. Подготовить средствами InterBase и Delphi клиентское приложение *Автомобили*, для которого задана условная схема базы данных (рис. 7.60) и необходимо получать информацию следующего плана:

   - определить автомобили данного года выпуска, марка которых содержит буквы "С", "М" или "Б". Отсортировать по убыванию;
   - определить количество автомобилей для каждого владельца;
   - определить среднюю и максимальную цены автомобилей каждой марки для некоторого года выпуска;
   - определить, есть ли владельцы черных или белых Мерседесов 1996 г. выпуска, категория которых "В" или "С";
   - среди средних цен марок автомобилей найти максимальную цену;
   - определить владельцев заданного года получения прав, у которых имеются автомобили, цена которых больше средней цены всех автомобилей.

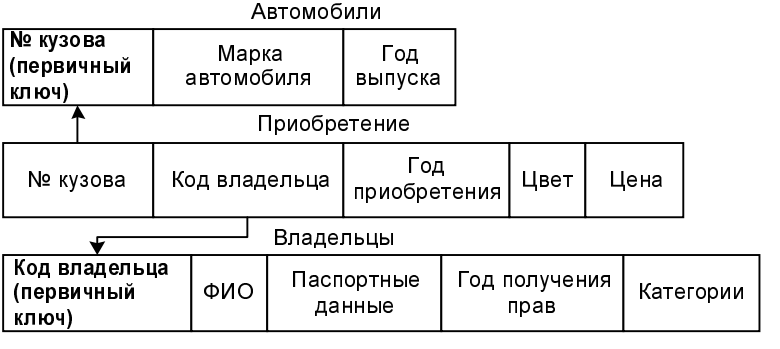


Рис. 7.60. Схема для базы данных "Автомобили"

9. Подготовить средствами InterBase и Delphi клиентское приложение *Грузоперевозки*, для которого задана условная схема базы данных (рис. 7.61) и необходимо получать информацию следующего плана:

   - получить список всех технических средств (вывести также технические характеристики), приобретенные за последние 5 лет;
   - получить список водителей, которые имеют категории "C", "C" и "D";
   - вывести список перевозок за определенную дату;
   - выполнить фактический расчет бензина для автомобилей, которые осуществили перевозки, на конкретную дату;
   - получить данные по расходованию бензина: дата, машина, отклонение в расходовании бензина (`расход по норме - (расход бензина / пробег машины ) * 100`);
   - суммарный пробег конкретной машины.

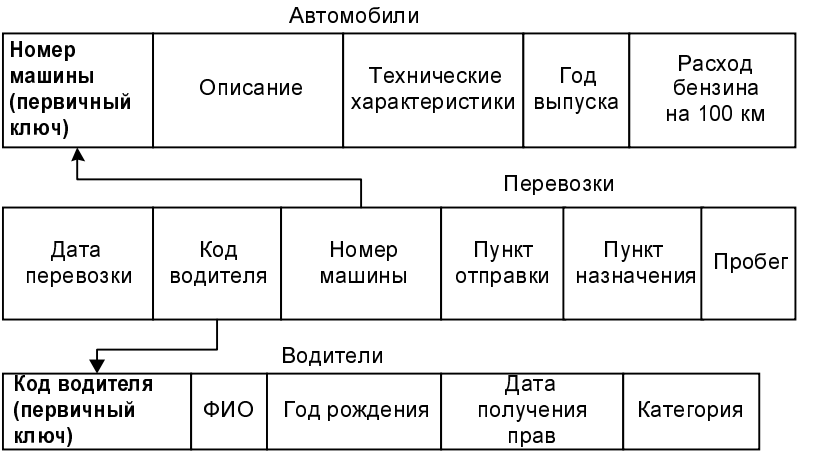


Рис. 7.61. Схема для базы данных "Грузоперевозки"

10. Подготовить средствами InterBase и Delphi клиентское приложение *Компьютерные игры*, для которого задана условная схема базы данных (рис. 7.62) и необходимо получать информацию следующего плана:

    - определить игры, выпущенные одной фирмой за определенный срок, отсортированные по возрастанию;
    - определить количество игр на дисках;
    - общую стоимость дисков за каждый год;
    - найти самую популярную игру;
    - определить диски, тираж которых меньше среднего по всей базе и цена отличается на 30% от средней цены по базе;
    - определить количество записей (на скольких дисках записана игра) для каждой фирмы, отсортированных по убыванию.

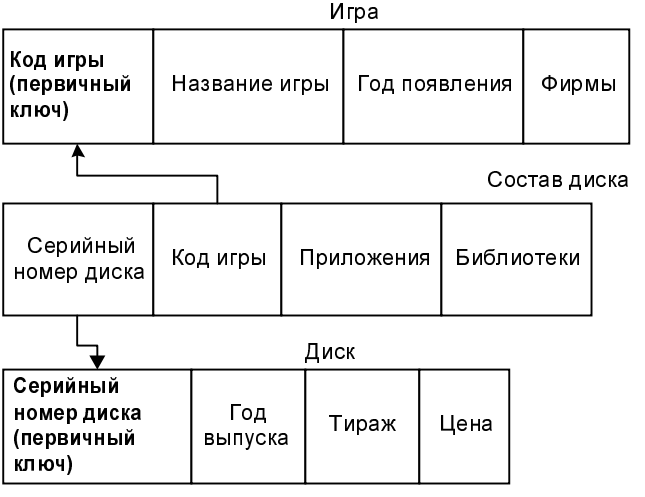


Рис. 7.62. Схема для базы данных "Компьютерные игры"

11. Подготовить средствами InterBase и Delphi клиентское приложение *Спортивный клуб*, для которого задана условная схема базы данных (рис. 7.63) и необходимо получать информацию следующего плана:

- получить списки участников по всем секциям, отсортированные по убыванию названия секции;
- определить общую плату по всем секциям;
- определить максимальную оплату за секцию;
- определить участников первого разряда, отсортированных по убыванию фамилий;
- определить всех участников секции данного тренера;
- определить общую плату участников для тех, которые посещали несколько секций;
- определить оплату за посещение секций на конкретную дату.

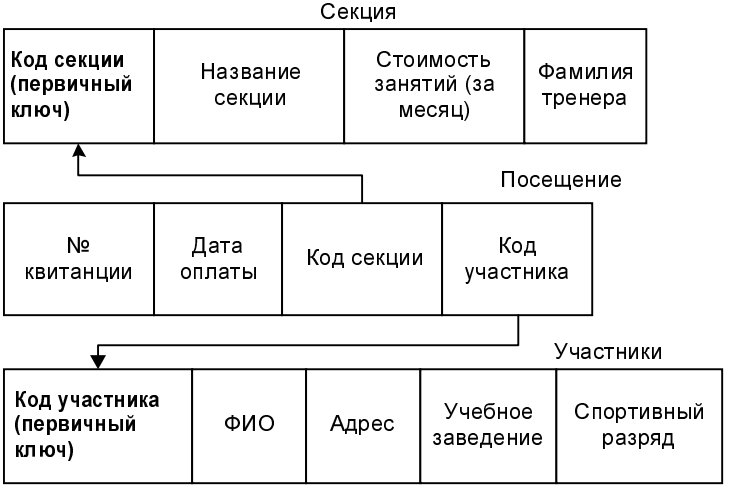


Рис. 7.63. Схема для базы данных "Спортивный клуб"

Рекомендуемая литература

1. Астахова И. Ф., Толстобров А. П., Мельников В. М. SQL в примерах и задачах: Учеб. Пособие. — Мн.: Новое знание, 2002.
2. Гарсиа-Молина Г., Ульман Д., Уидом Д. Системы баз данных. Полный курс.: Пер. с англ. — М.: Издательский дом "Вильямс", 2003.
3. Гетц К., Литвин П., Гилберт М. Access 2000. Руководство разработчика. — Киев: BHV, 2000.
4. Грофф Дж., Вайнберг П. SQL: Полное руководство: Пер. с англ. — 2-е изд., перераб. и доп. — Киев: Издательская группа BHV, 2001.
5. Дворжецкий А. В. SQL: Structured Query Language (Структурированный язык запросов). — М.: Познавательная книга плюс, 2001.
6. Дейт К. Дж. Введение в системы баз данных, 6-е изд.: Пер. с англ. — Киев; М.; СПб.: Издательский дом "Вильямс", 2000.
7. Дженнингс Р. Microsoft Access 97 в подлиннике. — СПб.: BHV-Петербург, 1999.
8. Дженнингс Р. Использование Microsoft Access 2000. Специальное издание.: Пер. с англ.: Учеб. пособие. — М.: Издательский дом "Вильямс", 2000.
9. Дейв Э., Йен С. Oracle. Проектирование баз данных: Пер. с англ. — Киев: Издательская группа BHV, 2000.

10. Карпова Т. С. Базы данных: модели, разработка, реализация. — СПб.: Питер, 2001.

11. Киммел П. Освой самостоятельно программирование для Microsoft Access 2000 за 24 часа. — М.: Вильямс, 2000.

12. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ.: Учеб. пособие. — М.: Издательский дом "Вильямс", 2000.

Предметный указатель

A

ACID-свойства 16, 17, 190, 191

B

BDE Administrator 400

I

IME 252
InterBase 377
  выбор текущего сервера и базы данных 384
  соединение с базой данных 383
  соединение с сервером 378
  создание базы данных 380

M

MEMO 262
Microsoft Access 227, 230

O

ODBC 232

Q

QBE 234

A

Авторизация пользователей 209
Архитектура "клиент-сервер" 20, 370, 371, 373
Атрибут 35, 40
Аудиторская проверка 211
Аутентификация 209

Б

База данных 5, 34
  безопасность 212
  детализация 13
  защита 207
  опасность 208
  описание 13
  содержание 13
  состояние 13
  схема 34, 104
Блокировка 197
  всей базы данных 199
  жесткая 200
  интервал 205
  на уровне страниц 200
  на уровне строк 200
  на уровне таблиц 199
  нежесткая 200
  параметр 204
  правила 200
  явная 202

B

Внешние соединения 78, 79
Восстановление 189, 193, 195
Выражение 96, 264

Г

Генератор 171
Группировка 339

Д

Дамп 196
Данные 5, 13
  выборка 127

(окончание рубрики см. на с. 484)

Данные *(окончание)*:
добавление 123
интеграция 21
мультимедиа 21
обновление 125
отбор 280
сортировка 280
тип 35
удаление 124
Домен 35, 39, 94

Ж

Журнал транзакций 193

З

Запрос 15, 45, 127, 234, 283
запуск 291
на выборку 297
на добавление 311
на обновление 314
на создание таблицы 310
на удаление 312
перекрестный 316
подчиненный 144
правила выполнения 143

И

Идентификатор 266
Имя поля 247
Индекс 14, 48, 51, 52, 117, 257
Инструкция SQL 88
ALTER TABLE 114
CREATE ASSERTION 116
CREATE INDEX 118
CREATE TABLE 106
CREATE VIEW 120
DELETE 124
DROP TABLE 114
DROP VIEW 120
FOREIGN KEY 108
INSERT 123
PRIMARY KEY 108
SELECT 127
SET CONSTRAINTS 110
UPDATE 125
CONSTRAINT 106
Информация 5

К

Каскадное обновление 261
Каскадное удаление 261
Кластеризация 51
Клиент-серверная архитектура 17
Ключ 35, 46, 47
Ключевое поле 257
Ключевое слово 88
Кнопки операторов 269
Команда 88
Комментарии 122
Константа 95, 266
Конструктор 331
запросов 287
макросов 345
отчетов 337, 338
таблиц 245, 247
форм 323, 324
Контрольный след 216
Корректность 166
Критерии отбора 294

М

Макрос 237, 344, 345
Маска ввода 254
Метаданные 13
Модель реляционная 33, 37
Модуль 237

Н

Независимость от данных 13

О

Объединение:
внутреннее 140
левое внешнее 141

операция UNION 138
перекрестное 140
по неравенству 140
по равенству 139
полное внешнее 141
правое внешнее 141
расширенный запрос 142
рекурсивное 140
Ограничение 20, 166
DEFERRABLE 110
INITIALY DEFERRED 110
INTITALY IMMEDIATE 110
NOT DEFERRABLE 110
атрибута 167
базы данных 167
домена 166
механизм проверки 110
таблицы 167
Окно базы данных 230
Оператор:
AND 133
COMMIT 191, 192
CROSS JOIN 140
FULL JOIN 141
GRANT 218
INNER JOIN 140
IS 134
LEFT JOIN 141
LOCK TABLE 202
MATCH FULL 108
MATCH PARTIAL 108
NOT 133
OR 132
REVOKE 220
RIGHT JOIN 141
ROLLBACK 191, 192
SAVEPOINT 192
UNION JOIN 142
переименования атрибутов 58
Операции реляционной алгебры:
выборка 65
вычитание 62
декартово произведение 62
деление 74
естественное соединение 73
объединение 59
пересечение 61
проекция 66
соединение 67, 68
тэта-соединение 68
экви-соединение 71
Определитель NULL 48
Оптимизатор 145
Отказ:
носителей 193, 195
системы 193, 194
Отношение 34, 39
атрибут 34
базовое 35, 44
выражаемое 45
заголовок 34, 40
именованное 44
кортеж 35
мощность 35, 40
неименованное 57
переменная 44
производное 35, 44
совместимое по типу 58
степень 35, 40
схема 34, 41
тело 34, 40
хранимое 46
экземпляр 41
Отчет 235, 338, 339

П

Панель:
аргументов 345
макрокоманд 345
элементов 326
Параллелизм 196
Параметры запуска 350
Первичный ключ 256
Поддержка целостности 210
Подстановочный знак 131
Поле выражения 268
Построитель выражений 268, 291, 292

Правило обновления 109
Правило удаления:
CASCADE 109
RESTRICT 109
SET DEFAULT 109
SET NULL 109
Предикат 46
ALL 158
ANY 157, 158
EXISTS 156, 157
IN 155
Предложение 88
FROM 128
GROUP BY 136
HAVING 137
ORDER BY 137
SELECT 127
WHERE 128
Представление 118, 209
Принцип избыточности 189
Проблема:
несовместимого анализа 197
промежуточных данных 197
пропавшего обновления 197

P

Резервное копирование 210
Реляционная модель данных 34
Реляционная целостность данных 46
Реляционные ограничения целостности 48

C

Свертка 50
Свойства:
запроса 299
поля 246, 247, 250
таблицы 274
Связь 36, 43, 260, 278
Сервер 17
баз данных 371
приложений 373
Сериализация 196
Система:
информационная 5
информационно-управляющая 5
обработки данных 3
с базой данных 5
хост 7
Системный каталог 121
Снимки 45
Соединение:
внешнее 285
внутреннее 285
по отношению 286
рекурсивное 285
Создание:
запросов 287
отчетов 335
таблиц 244, 274
форм 321
Сортировка 280, 339
Средства контроля 208, 211
Столбец, определение 106
Страница доступа к данным 235
Структура данных 37
СУБД 13, 18, 19
Схема базы данных 11, 43
Схема данных 233, 244, 259, 277
Сцепление 63
Счетчик 263

T

Таблица 233
изменение 114
псевдоним 117
создание 106
удаление 114
Технология:
COM 374
CORBA 374
Тип данных 89, 247
пользовательский 93
Точка:
контрольная 194

отката 192
синхронизации 192
Транзакция 189
протокол двухфазной блокировки 206
упорядоченность 205
Трехуровневая архитектура 9
Триггер 20, 168, 169
Тупиковая ситуация 201

У

Управление доступом:
избирательное 213
обязательное 217
Уровень архитектуры БД 10, 11
Уровень изоляции 203
READ COMMITTED 203
READ UNCOMMITTED 204
REPEATABLE READ 203
SERIALIZABLE 203
Условие отбора:
принадлежность диапазону 129
равнество значению NULL 132
соответствие шаблону 131
составное 133
сравнение 129
членство во множестве 130
Условия на значения 109
Утверждение 116

Ф

Файл индексный 49
Фильтр 281, 282
Форма 234
заголовок 326
кнопка системного меню 331
колонтитулы 326
область данных 325
примечание 326
создание 331
Функция 186, 266
встроенная 96
для работы с датами 103
для работы со строками 97
математическая 102
преобразования 102
статистическая 134, 135

Х

Хеширование 52
Хеш-код 49
Хранимая процедура 172, 173, 174

Ц

Целостность 165
корпоративные ограничения 48
отношений 48
по ссылкам 167
ссылочная 48

Ш

Шаблон 131
Шифрование 210

Э

Элемент управления 326, 329
Элементы выражения 269

Я

Язык:
DDL 84, 103
DML 85, 123
DQL 85, 127
SQL 83, 84, 87
администрирования данных 86
управления транзакциями 8