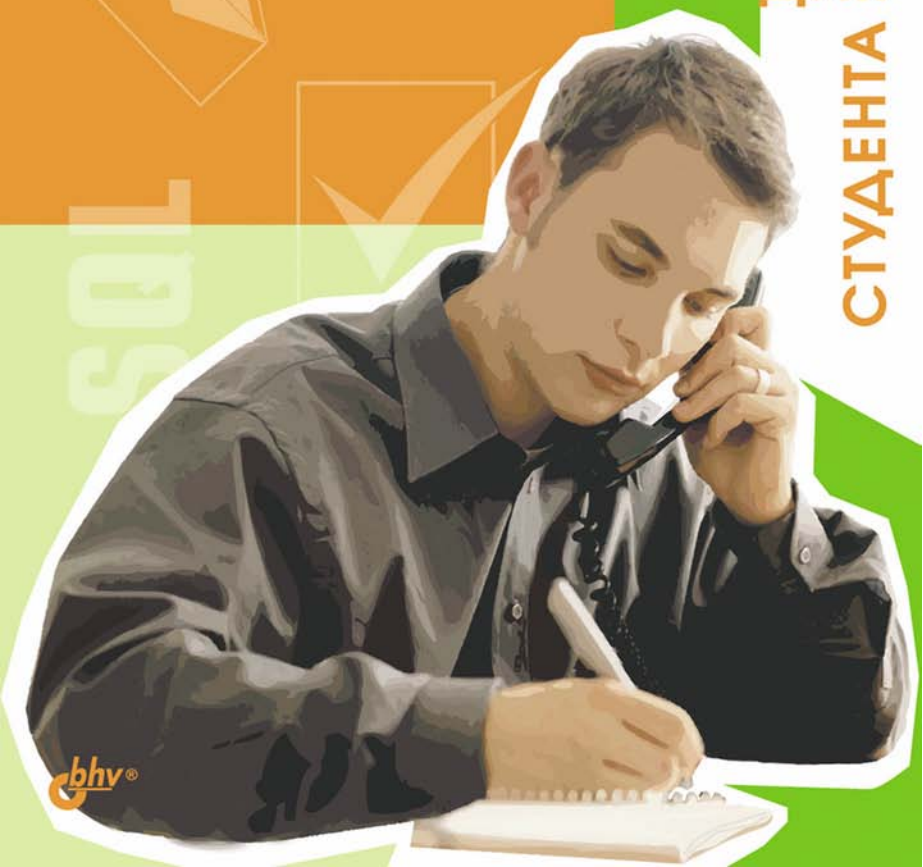
В. В. Дунаев

БАЗЫ ДАННЫХ
ЯЗЫК SQL
ДЛЯ СТУДЕНТА



bhv®

Вадим Дунаев

БАЗЫ ДАННЫХ
ЯЗЫК SQL
ДЛЯ СТУДЕНТА

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.068+800.92SQL
ББК 32.973.26-018.1
Д83

Дунаев В. В.

Д83 Базы данных. Язык SQL. — СПб.: БХВ-Петербург, 2006. — 288 с.: ил.

ISBN 978-5-94157-823-8

Рассмотрен язык структурированных запросов для взаимодействия с базами данных — SQL— начиная с доступного изложения теории отношений (реляционной теории) и заканчивая вопросами администрирования СУБД с помощью запросов. На практических примерах подробно описаны основные конструкции языка, а также различные типы запросов: простые, сложные, рекурсивные. Показано, как осуществлять вычисления в запросах с помощью агрегатных функций и условных выражений. Рассмотрены операции над наборами записей, соединение таблиц, модификация данных, транзакции, курсоры, хранимые процедуры и функции. Уделено внимание администрированию СУБД с помощью запросов. Материал сопровождается задачами для самостоятельного решения.

Для студентов и программистов

УДК 681.3.068+800.92SQL
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.02.06.
Формат $60\times90^{1}/_{16}$. Печать офсетная. Усл. печ. л. 18.
Тираж 3000 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 978-5-94157-823-8

Оглавление

От автора

Занимаясь приложениями реляционных баз данных более десяти лет, после двухлетней паузы я вдруг неожиданно для себя обнаружил, что изрядно подзабыл язык SQL. В памяти остались лишь общие идеи. Пришлось обратиться к нескольким книгам. И тогда я подумал, что хорошо бы иметь под рукой небольшую книжку, в которой было бы все самое необходимое, чтобы легко и быстро все вспомнить. А тем, кто не знал SQL никогда, эта книжка могла бы помочь быстро его освоить в такой степени, чтобы получать практически полезные результаты. Воплощение этой идеи вы держите в руках. Надеюсь, что данная книга окажется полезной как для новичков, так и для успевших кое-что забыть.

Благодарности

Я искренне благодарен коллегам Дмитрию Лямаеву и Игорю Слюсаренко за практические советы и консультации по техническим вопросам.

Введение

SQL (Structured Query Language) — это структурированный язык запросов к реляционным базам данных. На этом языке можно формулировать выражения (запросы), которые извлекают требуемые данные, модифицируют их, создают таблицы и изменяют их структуры, определяют права доступа к данным и многое другое.

Запросы выполняются системой управления базой данных (СУБД). Если вы не являетесь специалистом по разработке и администрированию баз данных, то вполне можете быть их пользователем, который просматривает или/и изменяет данные в уже имеющихся таблицах. Во многих случаях эти и другие операции с базой данных выполняются с помощью специальных приложений, предоставляющих пользователю удобный интерфейс. Обычно приложения пишутся на специальных языках программирования (C, Pascal, Visual Basic и т. п.) и чаще всего создаются с помощью интегрированных сред разработки, например, Delphi, C++ Builder и др. Однако доступ к базе данных можно получить и без них — с помощью только SQL. Замечу также, что и специализированные приложения обычно используют SQL-фрагменты кода при обращениях к базе данных.

Таким образом, SQL — широко распространенный стандартный язык работы с реляционными базами данных. Синтаксис этого языка достаточно прост, чтобы его могли использовать рядовые пользователи, а не только программисты. В настоящее время обычный пользователь компьютера должен владеть, по крайней мере, текстовым редактором (например, Microsoft Word) и электронными таблицами (например, Microsoft Excel). Неплохо, если

он также умеет пользоваться базами данных. Различных СУБД существует много, а универсальное средство работы с базами данных одно — SQL. Знание SQL, хотя бы его основ, и умение его применять для поиска и анализа данных является фундаментальной частью компьютерной грамотности даже рядовых пользователей.

Как устроена эта книга

Данная книга посвящена языку манипулирования реляционными базами данных, а не их проектированию и сопровождению. Однако успех изучения и применения SQL существенно зависит от понимания того, как устроена реляционная база данных. Теоретическим источником реляционных баз данных является теория отношений (реляционная теория). Основам этой теории посвящена *глава 1*. Несмотря на обилие символики, изложенный в ней материал достаточно прост и вполне доступен широкому кругу читателей. Однако те, кого интересуют непосредственно SQL и возможность быстрее получить практический результат, могут пропустить эту главу при первом чтении. Изложение последующих глав построено в основном на рассмотрении примеров.

Таблицы, из которых состоит любая реляционная база данных, представляют собой некоторые отношения, а отношения являются не чем иным, как множествами записей (строк). Все запросы к базе данных, направленные на извлечение из нее нужных сведений, интерпретируются как инструкции по выполнению тех или иных операций, являющихся в конечном счете операциями алгебры множеств. Более или менее серьезные базы данных состоят из нескольких таблиц, между которыми могут существовать связи. Рассмотрение в реляционной теории декомпозиции одной таблицы на несколько других позволит вам понять противоположный процесс — проектирование базы данных как композиции нескольких таблиц.

Главы 2—4 содержат минимальный набор сведений, позволяющий в большинстве случаев извлекать из базы данных необходимую информацию.

В *главе 2* кратко рассматривается история и составные части SQL, а также типы данных. При первом прочтении желательно

получить хотя бы поверхностное представление о типах данных. Дело в том, что наборы типов данных SQL и типов столбцов таблиц, поддерживаемых той или иной СУБД, как правило, не совпадают. Однако между большинством из них есть соответствие. Кроме того, можно использовать функцию приведения к заданному типу.

В *главе 3* описываются запросы на выборку данных, которые наиболее часто используются на практике. Это так называемые простые запросы, не содержащие вложенных запросов. Здесь же рассматриваются условия, которые могут применяться не только в запросах на выборку, но и при модификации данных.

Глава 4 посвящена вычислениям в запросах с помощью итоговых (агрегатных) функций и условных выражений.

В *главе 5* рассматриваются так называемые сложные запросы, а также теоретико-множественные операции над наборами записей, соединения таблиц и рекурсивные запросы. Нередко для получения данных приходится вначале выполнить некий вспомогательный запрос, чтобы затем его результат использовать при формулировке условия выборки данных в основном запросе. Вспомогательный запрос включается в выражение основного запроса, который называют сложным (содержащим подзапрос). Теоретико-множественные операции позволяют из нескольких наборов записей получить другой набор записей — объединение, пересечение или разность исходных наборов. Операция соединения таблиц дает в результате таблицу, записи в которой получаются путем некоторой комбинации записей соединяемых таблиц.

Модификация (изменение, добавление и удаление) данных описывается в *главе 6*.

Рано или поздно возникает задача создания новых или изменения структуры уже существующих таблиц. Средства SQL, позволяющие это сделать, рассмотрены в *главе 7*. Здесь же описываются представления — виртуальные таблицы, доступные многим пользователям.

При выполнении многоэтапных операций с базами данных, особенно в многопользовательском режиме, последовательности SQL-выражений объединяются в транзакции. Если какое-либо SQL-выражение в транзакции по какой-то причине не выполнилось, отменяется действие всех SQL-выражений в этой транзакции,

а база данных возвращается в исходное состояние, в котором она находилась до начала транзакции. Иначе говоря, выполняется принцип "все или ничего". Транзакции описываются в *главе 8*.

В *главе 9* рассмотрены так называемые курсоры. Курсор позволяет сделать обработку одной или нескольких записей таблицы с помощью SQL-выражений и, таким образом, упрощает совместное использование SQL с другими языками программирования, на которых пишутся приложения баз данных. Выборка записей производится с помощью SQL-запроса, а проверка их содержимого — с помощью кода на процедурном языке.

Глава 10 посвящена постоянно хранимым модулям. На языке SQL можно написать процедуры и функции, которые будут содержать объявления переменных и составные команды SQL. Коды этих процедур и функций могут содержать управляющие структуры, а также обычные SQL-выражения. Функции и процедуры задаются в рамках так называемого модуля, который представляет собой контейнер для их размещения. Модуль создается специальной командой SQL и сохраняется в метаданных базы данных, т. е. становится ее компонентом, подобно таблицам, индексам и т. д. Таким образом, однажды создав модуль с процедурами и функциями, вы можете затем в частных SQL-запросах использовать их вызовы.

В *главе 11* рассматриваются некоторые возможности SQL по администрированию базы данных, а именно, по предоставлению прав доступа к ее объектам. Разграничение прав доступа является важным средством защиты базы данных от неправильного использования содержащейся в ней информации различными категориями пользователей.

В конце некоторых глав приводятся задачи для самоконтроля усвоения прочитанного материала. Список зарезервированных слов SQL приведен в приложении.

Как выполнять примеры SQL-выражений

Чтобы выполнить примеры и решить задачи, приведенные в книге, необходимо установить на компьютере какую-нибудь СУБД, которых существует множество. Наиболее простым вариантом является Microsoft Access. Однако не все возможности SQL-92

поддерживаются этой СУБД. Я рекомендую установить какой-нибудь SQL-сервер, например, Microsoft SQL Server 2000 или PostgreSQL. Далее мы кратко рассмотрим создание базы данных и SQL-выражений с помощью SQL Server 2000 и Access 2003.

Работа с Microsoft SQL Server 2000

Установка Microsoft SQL Server 2000

Процедура инсталляции SQL Server 2000 с установочного CD-диска довольно проста. Если инсталлятор не запустился автоматически, то следует запустить программу \x86\setup\setupsql.exe, расположенную на установочном CD. Далее мастер предоставит вам возможность указать параметры установки. Один из возможных вариантов установки, пригодный для большинства новичков, работающих на локальном компьютере, выглядит следующим образом:

1. **Local Computer** (Локальный компьютер).
2. **Server and Client Tools** (Инструменты сервера и клиента). При этом будет инсталлирован сам сервер и средства администрирования.
3. Настройка учетных записей служб:
   - **Use the same account for each service. Auto start SQL Server Service** (Использовать одну учетную запись для всех служб с их автоматическим запуском при загрузке операционной системы);
   - **Use the Local System account** (Использовать локальную учетную запись);
   - **Auto Start Service** (Автоматический запуск служб при загрузке операционной системы).
4. **Typical** (Установка всех компонентов).

После инсталляции сервера баз данных в меню Windows **Пуск | Программы | Microsoft SQL Server** запустите утилиту Enterprise Manager, которая предоставляет пользовательский интерфейс для настройки, управления и доступа к данным на сервере (рис. В1). В левой части окна утилиты отображается древовидный список, в котором щелчком левой кнопкой мыши следует

раскрыть узел **Microsoft SQL Servers/SQL Server Group**. Если установка прошла успешно, то в этом узле должен находиться узел, имя которого состоит из имени вашей учетной записи, за которым следует "(Windows NT)". На рис. В1 это **DVV (Windows NT)**. В правой части окна отображаются элементы того узла, который выделен в левой части окна (в дереве).

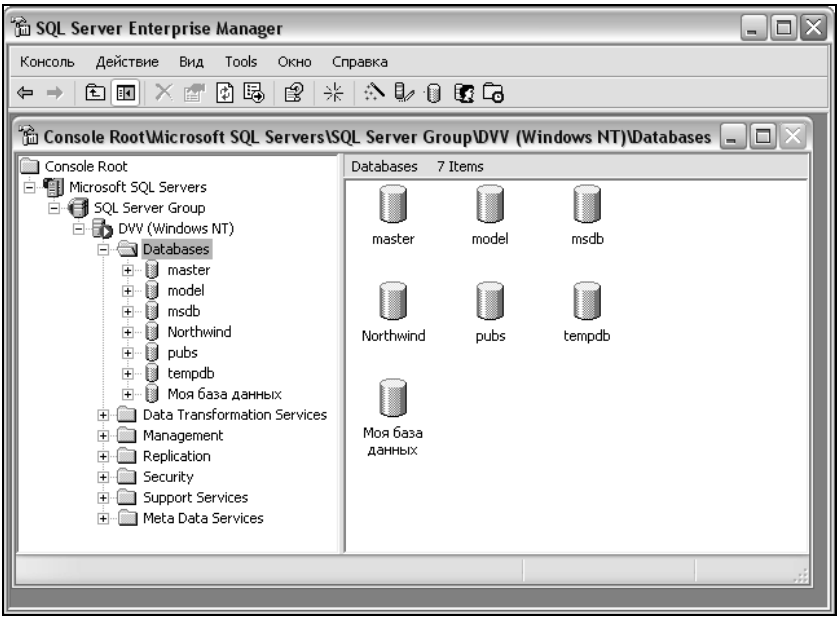


Рис. В1. Окно утилиты Enterprise Manager

Создание базы данных в Enterprise Manager

Чтобы выполнить учебные примеры SQL, необходимо создать учебную базу данных, на которой вы будете экспериментировать. Для этого в древовидном списке окна утилиты Enterprise Manager раскройте узел, соответствующий вашей учетной записи (на рис. В1 **DVV (Windows NT)**). Среди содержащихся в нем подузлов имеется **Databases** (Базы данных). Щелкните на нем правой кнопкой мыши и в раскрывшемся контекстном меню выберите **New Database** (Новая база данных). В результате откроется диалоговое окно **Database Properties** (Свойства базы данных).

В поле **Name** (Имя) введите имя базы данных (например, "Моя база данных") и щелкните на кнопке **OK**. В результате в узле **Databases** появится узел с именем, совпадающим с именем базы данных.

Итак, создана пустая база данных, которая пока не содержит ни одной таблицы с пользовательскими данными. Теперь создайте какую-нибудь таблицу, например, `Клиенты` с символьными столбцами `Имя`, `Адрес`, `Телефон`. Для этого щелкните правой кнопкой мыши на имени вашей базы данных в древовидном списке и в контекстном меню выберите **Создать | Table**. В результате откроется окно **New Table** (Новая таблица), показанное на рис. В2. В этом окне следует ввести имена столбцов (**Column Name**), тип данных (**Data Type**) и длину данных (**Length**). В рассматриваемом примере все столбцы являются символьными (строковыми, текстовыми), т. е. имеют тип `char`.

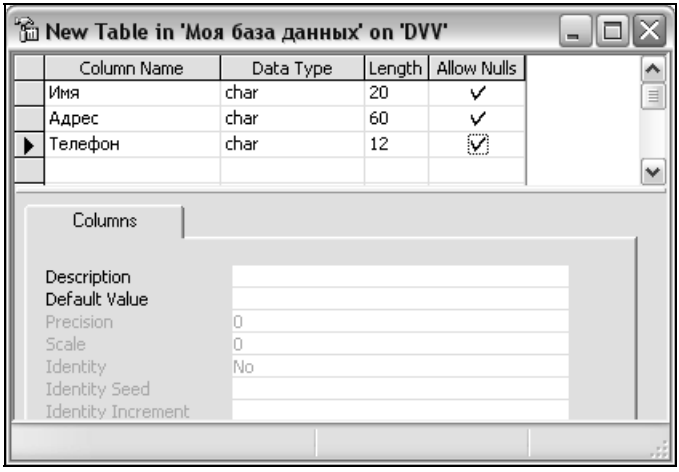


Рис. В2. Окно **New Table** для определения столбцов таблицы

После завершения определения столбцов (структуры) создаваемой таблицы закройте окно, после чего появится запрос о сохранении введенных данных. Подтвердите необходимость сохранения и в открывшемся диалоговом окне введите имя созданной таблицы (например, `Клиенты`), а затем щелкните на кнопке **OK**.

Если в древовидном списке раскрыть узел с именем вашей базы данных и щелкнуть на узле **Tables** (Таблицы), то в правой части

окна Enterprise Manager отобразятся все таблицы, входящие в эту базу данных. Среди множества служебных таблиц вы найдете и свою только что созданную таблицу.

Вы можете изменить структуру созданной таблицы или/и изменить содержащиеся в ней данные.

Для изменения структуры таблицы щелкните правой кнопкой мыши на ее имени и в контекстном меню выберите **Design Table** (Разработка таблицы). В результате откроется окно, аналогичное показанному на рис. В2. Теперь можно добавить новые или удалить имеющиеся столбцы, изменить их имена и/или другие параметры.

Для ввода, редактирования и удаления данных в таблице щелкните правой кнопкой мыши на ее имени и в контекстном меню выберите **Open Table | Return all rows** (Открыть таблицу | Вернуть все записи). В результате откроется окно, показанное на рис. В3. В этом окне можно вводить и изменять значения столбцов, добавлять и удалять строки (записи). Поскольку новая таблица пуста, то начать следует с добавления записей.

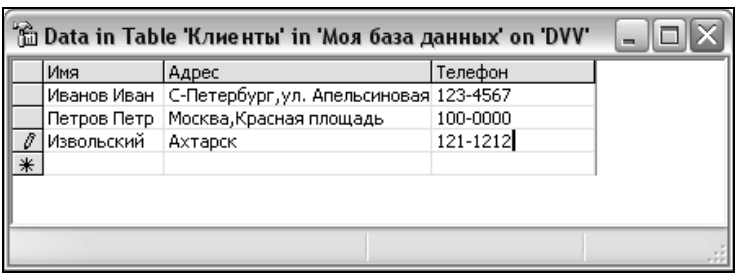


Рис. В3. Окно для модификации данных таблицы

Аналогичным образом можно создать и другие таблицы базы данных.

Создание и выполнение SQL-выражений в SQL Query Analyser

Для создания, редактирования и выполнения выражений на языке SQL рекомендуется использовать утилиту SQL Query Analyser. Эту утилиту можно вызвать из меню **Tools** (Инструменты),

расположенного в верхней части окна Enterprise Manager. В результате откроется окно, показанное на рис. В4. Здесь можно ввести, как в обычном текстовом редакторе, SQL-выражение.

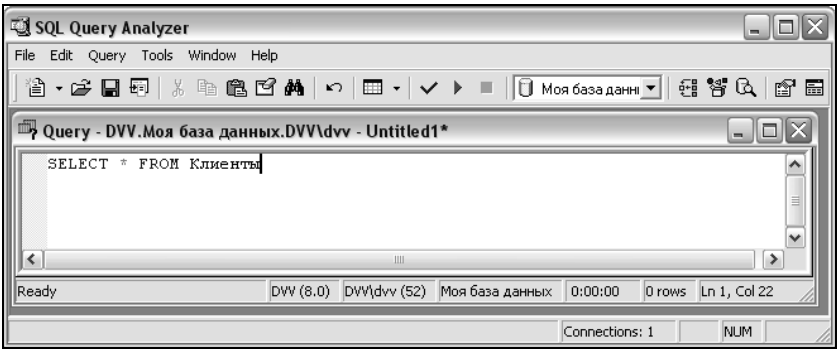


Рис. В4. Окно утилиты SQL Query Analyser

Прежде чем выполнить введенное SQL-выражение, рекомендуется его проверить, щелкнув на кнопке **Parse Query** (Анализировать запрос) с изображением галочки или нажав клавиши <Ctrl>+<F5>. Для выполнения выражения достаточно щелкнуть на кнопке с изображением стрелки вправо или нажать клавишу <F5>.

При желании созданное SQL-выражение можно сохранить в текстовом файле с расширением sql. Сохраненные в файлах SQL-выражения можно снова открыть в окне SQL Query Analyser. Соответствующие команды выбираются из меню **File**.

Работа с Microsoft Access 2003

Создание базы данных

При запуске программы Access открывается главное окно, в правой части которого находится панель **Приступая к работе**. На этой панели в разделе **Открыть** выберите команду **Создать файл**. В результате отобразится панель **Создание файла**. В разделе **Создание** на этой панели выберите команду **Новая база данных**. В открывшемся диалоговом окне **Файл новой базы данных** введите имя файла создаваемой базы данных (например, "Моя база

данных") и сохраните его в какой-нибудь папке. Имя файла базы данных в Access имеет расширение mdb.

Итак, пустая база данных создана. В главном окне Access откроется окно **Моя база данных: база данных**. В этом окне выберите **Создание таблицы в режиме конструктора**. Откроется окно **Таблица1: таблица**, показанное на рис. В5. В этом окне определяется структура (свойства столбцов) таблицы — имена, типы, размеры и другие параметры столбцов.

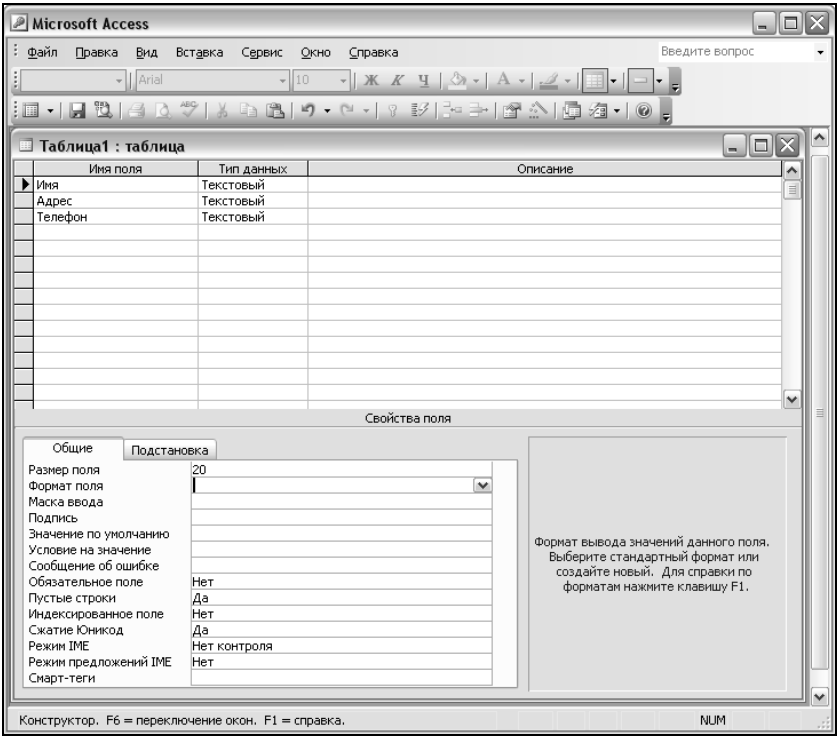


Рис. В5. Окно **Таблица1: таблица** для определения столбцов таблицы

По завершении определения столбцов (структуры) создаваемой таблицы закройте это окно, после чего появится запрос о сохранении введенных данных. Подтвердите сохранение и в открывшемся диалоговом окне введите имя созданной таблицы (например, `Клиенты`) и щелкните на кнопке **ОК**.

В окне **Моя база данных: база данных** появится пиктограмма с именем созданной таблицы. Вы можете изменить структуру созданной таблицы или/и изменить содержащиеся в ней данные.

Для изменения структуры таблицы щелкните правой кнопкой мыши на ее имени и в контекстном меню выберите **Конструктор**. В результате откроется окно, аналогичное показанному на рис. В5. Теперь можно добавлять новые или удалять имеющиеся столбцы, изменять их имена и/или другие параметры.

Двойной щелчок на пиктограмме с именем таблицы откроет ее в режиме редактирования данных (рис. В6). В этом окне можно вводить и изменять значения столбцов, добавлять и удалять строки (записи). Поскольку новая таблица пуста, то следует начать с добавления записей.

Рис. В6. Окно для модификации данных таблицы

Аналогичным образом можно создать и другие таблицы базы данных.

Создание и выполнение SQL-выражений

В окне **Моя база данных: база данных** (см. рис. В6) в списке объектов выберите **Запросы**, а в области задач, расположенной в правой части этого окна, выполните двойной щелчок на опции **Создание запроса в режиме конструктора**. В результате откроются два окна: **Запрос1: запрос на выборку** и **Добавление таблицы**. Окно **Добавление таблицы** закройте, после чего на экране останутся окна, показанные на рис. В7.

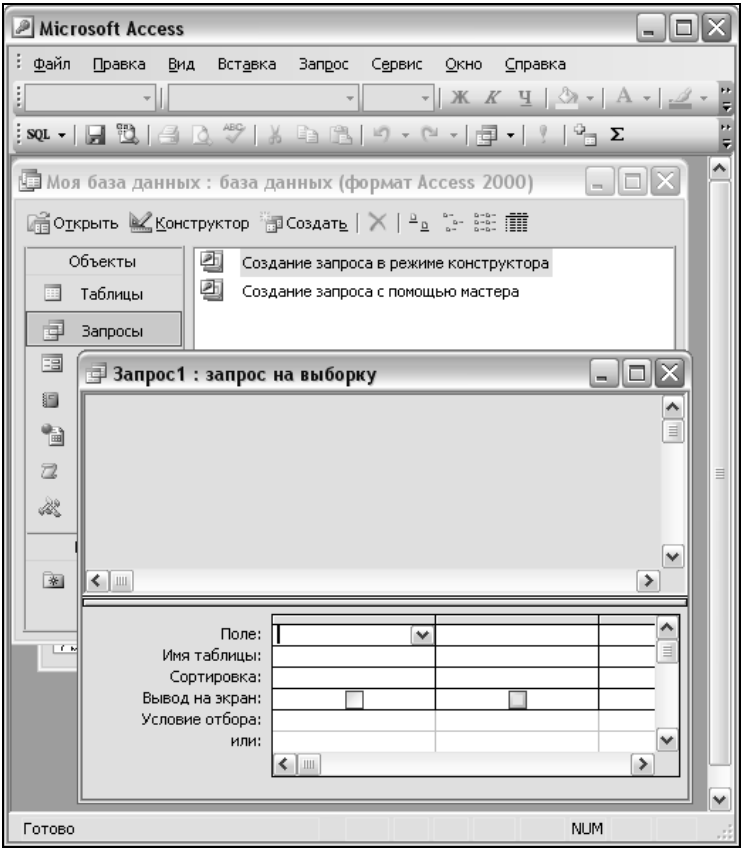


Рис. В7. Окно для ввода SQL-выражений в режиме конструктора

В меню **Вид** главного окна Access выберите команду **Режим SQL** или щелкните правой кнопкой мыши на верхней области окна **Запрос1: запрос на выборку** и в раскрывшемся контекстном меню выберите **Режим SQL**. В результате окно запросов примет вид, как показано на рис. В8. Это текстовый редактор, в котором можно вводить и редактировать выражения на языке SQL. При создании нового запроса в области ввода появляется строка "`SELECT;`", т. е. ключевое слово запроса, на выборку данных. Если вам необходимо выбрать данные, то необходимо дописать недостающие элементы SQL-выражения. Для создания других запросов (например, модификации данных, создания таблиц и др.) следует удалить ключевое слово `SELECT` и написать вместо него требуемый оператор.

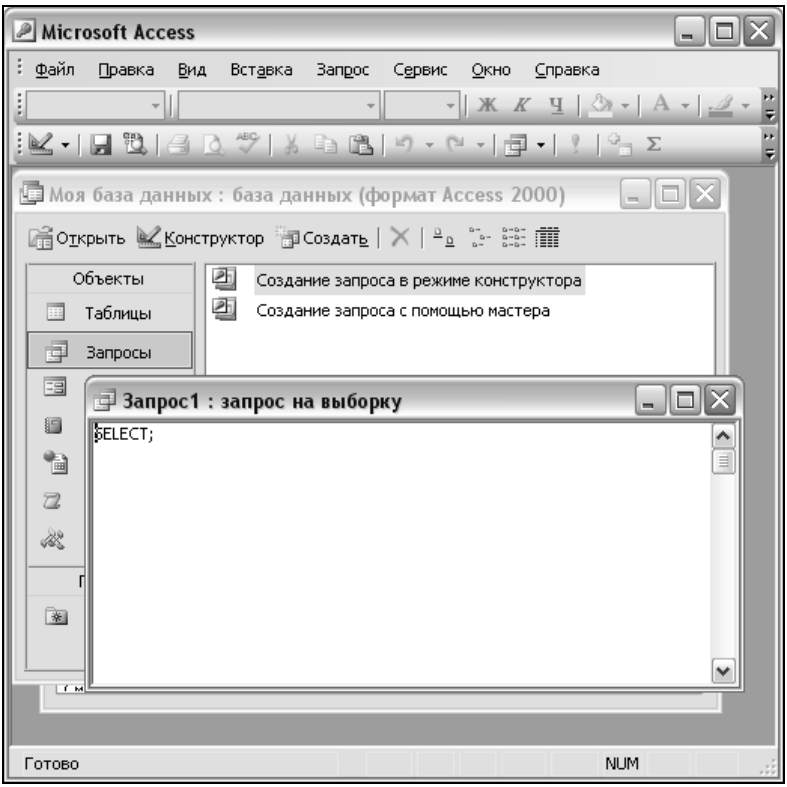


Рис. В8. Окно для ввода SQL-выражений в режиме SQL

Созданные запросы сохраняются в базе данных под именами, задаваемыми пользователем. При закрытии редактора запросов появляется предложение сохранить созданный запрос. В случае положительного ответа открывается диалоговое окно, в котором можно ввести имя запроса. Если выражение запроса было задано корректно, то его имя появится в списке запросов в окне **Моя база данных: база данных**. В противном случае появится сообщение об ошибке.

Чтобы отредактировать созданный ранее запрос, достаточно щелкнуть правой кнопкой мыши на его имени и в контекстном меню выбрать **Конструктор**.

Чтобы выполнить запрос, достаточно сделать двойной щелчок на его имени в списке запросов или в контекстном меню, раскрываемом щелчком правой кнопкой мыши на имени запроса, выбрать команду **Открыть**.

Глава 1

Основы реляционных баз данных

Программные средства работы с реляционными базами данных, в отличие от многих других, базируются на довольно строгой теории отношений, которая, в свою очередь, основана на теории множеств. Чтобы понять суть задачи создания реляционной базы данных, а также операций над данными, достаточно рассмотреть на теоретическом (абстрактном) уровне всего лишь несколько основных положений теории отношений. Это быстрее и, в конечном счете, лучше, как мне представляется, чем изучать большое количество конкретных примеров и ситуаций.

Таблицы, из которых состоит любая реляционная база данных, представляют собой некоторые отношения, а отношения являются не чем иным, как множествами. Все запросы к базе данных, направленные на извлечение из нее нужных записей, интерпретируются как инструкции по выполнению тех или иных операций, являющихся, в конечном счете, операциями алгебры множеств и исчисления предикатов.

В этой главе мы рассмотрим основные вопросы, относящиеся к реляционным базам данных, с более общей точки зрения. Зато все последующие главы будут посвящены конкретным приемам использования языка SQL.

1.1. Множества

Множество является настолько общим понятием, что не имеет определения, которое можно было бы выразить в еще более общих и простых понятиях. Поэтому в математической теории множеств определение (в математическом смысле) понятия

множества отсутствует. Вместе с тем, в его основе лежит некий образ, который Георг Кантор (один из создателей теории множеств) описал как собрание определенных и различных между собой объектов интуиции или интеллекта, мыслимое как единое целое. Эти объекты называются элементами множества.

Примечание

В математике многое не так, как в других науках. Если мы обратим внимание на теорию, скажем, "блям-блямчиков", то с точки зрения математической культуры совсем не важно, что такое "блям-блямчики". Иначе говоря, строгое в математическом смысле определение "блям-блямчиков" не входит в теорию. "Блям-блямчики" — лишь наименование предмета теории, а ее содержание — выявление свойств и отношений "блям-блямчиков". Только изучив эту теорию, мы получим более или менее полное представление о том, что такое "блям-блямчики". Это представление возникает из понимания того, каким образом объект изучения и его компоненты связаны между собой и объектами окружающего мира. Так, например, К. Поппер (исследователь логических аспектов языка) удачно заметил, что смысл слов определяется через их использование в речи (текстах). Тем не менее, любая книга по теории, в том числе и математической, обычно начинается с пространных описаний предмета и стремится дать более четкое его "определение". Однако это по существу лишь стимуляция интуиции читателя, направленная на то, чтобы он настроился на предмет изучения и преодолел возможные психологические барьеры, возникающие при встрече с новым. По большому счету стимуляция интуиции не имеет непосредственного отношения к самой теории. Другими словами, интерпретация теории не является частью этой теории. Это задача ее применения к жизни.

Существенным для канторовского понимания множества является то, что собрание объектов само по себе рассматривается как один объект. На природу объектов, которые могут входить в множество, не накладывается никаких ограничений. Это могут быть числа, наборы символов, люди, атомы и т. п.

Множества могут быть конечными и бесконечными. *Конечные* множества содержат элементы, которые можно сосчитать или перечислить. Это означает, во-первых, что имеется *принципиальная* возможность сопоставить каждому элементу множества некоторое натуральное число (1, 2, 3, ...), и, во-вторых, этот пересчет когда-нибудь закончится. Так, например, бесконечное множество

всех целых чисел можно начать перечислять, но этот процесс никогда не закончится: для любого целого числа можно создать следующее, прибавив к нему 1. А множество всех действительных чисел, также являющееся бесконечным, даже начать перечислять невозможно. Интересно, что этот факт был установлен Кантором только лишь в конце XIX века и произвел на математиков ошеломляющее впечатление.

Существуют и конечные множества, перечислить элементы которых *практически* невозможно. Я хочу обратить ваше внимание на слово "практически". Представителем таких множеств может служить, например, множество всех погибших в Куликовской битве, а также множество всевозможных последовательностей нулей и единиц длиной 100. Мы уверены, что эти множества конечны. Количество погибших не может быть бесконечным, но мы либо не знаем, как их перечислить, либо решение этой задачи требует неимоверно больших затрат ресурсов. Количество последовательностей из нулей и единиц длиной 100 равно 2^{100}. Это число больше количества атомов в видимой части вселенной, и, следовательно, не хватит жизни многих поколений людей, чтобы перечислить (или сгенерировать) такие последовательности даже с помощью самого быстродействующего компьютера. Однако в математике практическая и теоретическая невозможности чего-либо — это различные вещи. Практические трудности игнорируются математиками, а выявление принципиальной недостижимости (фундаментальных пределов) чего-либо является для них чрезвычайно ценным результатом. Математическая теория множеств рассматривается сейчас как фундамент всей математики, а также как пространство и средство изучения бесконечного. Это очень интересно, но данная книга не об этом.

В компьютерной практике и, в частности, в базах данных мы имеем дело с конечными множествами, хотя иногда и очень большими. Поэтому мы застрахованы от логических неопределенностей и тупиков, которые могут встретиться в области бесконечных множеств. Однако нам предстоит преодолевать практические трудности, связанные с очень большими конечными множествами, но это задача скорее технологии, чем математики. Теперь обратимся к формальным аспектам конечных множеств, чтобы кратко и ясно представить себе то, с чем в большом многообразии частностей мы имеем дело в жизни.

Итак, *множество* является тем, что имеет элементы, а *элементы* — это то, что либо входит, либо не входит в данное множество. В множестве не может быть двух или более одинаковых элементов, а порядок расположения элементов в множестве не имеет никакого значения. Заметим, что этим множества отличаются от массивов — объектов языков программирования и компьютерных программ. Массивы в программах так или иначе упорядочены и могут содержать одинаковые элементы. Вместе с тем, ничто не мешает рассматривать массивы как модель (некоторое приближение) множеств. Важно лишь помнить, что массив — это некоторое программное воплощение идеи конечного множества, имеющее дополнительные свойства, которые не обязано иметь множество.

Обозначим некоторое множество символом A, а все элементы, входящие в это множество, — символами $a_1, a_2, \ldots, a_N$. Тогда запись $A = \{a_1, a_2, \ldots, a_N\}$ означает, что множество A содержит только те элементы $a_1, a_2, \ldots, a_N$, которые указаны в фигурных скобках. Напомню, что все элементы различны, а их порядок в фигурных скобках не имеет значения.

Какой-либо элемент x может принадлежать или не принадлежать множеству A. Чтобы указать, что элемент x принадлежит множеству A, пишут $x \in A$, а если x не принадлежит A, то пишут $x \notin A$. Множество может и не содержать элементов, тогда его называют пустым и обозначают как $\varnothing$.

Простейший способ для определения конкретного множества состоит в том, чтобы явно указать все элементы, принадлежащие этому множеству. Это так называемый *экстенсиональный* способ определения множества. Если количество элементов, входящих в множество A, невелико, то этот способ вполне применим на практике: достаточно написать выражение вида $A = \{a_1, a_2, \ldots, a_N\}$.

Однако при достаточно больших множествах этот способ не подходит. Тогда используют так называемый *интенсиональный* (неявный) способ определения множеств. Он основан на использовании некоторой функции (алгоритма), которая определяет для каждого элемента, принадлежит ли он данному множеству или нет. Пусть для некоторого множества A определена такая функ-

ция $P_A(x)$. Если вместо переменной x подставить в ее выражение конкретный элемент a, то результатом вычисления $P_A(a)$ будет либо ИСТИНА (`true`), либо ЛОЖЬ (`false`), в зависимости от того, принадлежит или нет элемент a множеству A. Таким образом, функция $P_A(x)$ принимает в качестве аргументов элементы из некоторой области (универсума) и возвращает одно из двух значений, которые мы здесь обозначаем как ИСТИНА и ЛОЖЬ (т. е. функция $P_A(x)$ является двузначной). С другой стороны, мы можем сами определить какую-нибудь двузначную функцию $Q(x)$, принимающую в качестве аргумента значения из некоторого универсума. Тогда эта функция будет определять некоторое множество, а именно множество всех тех элементов универсума, для которых эта функция возвращает значение ИСТИНА. Указанные двузначные функции в математической логике называются *предикатами*.

В качестве примера рассмотрим выражение $x < 5$. Это типичное выражение сравнения. Здесь через x обозначена переменная, значения которой берутся из множества всех действительных чисел. Тогда это выражение может принимать значения ИСТИНА или ЛОЖЬ в зависимости от того, какое значение будет подставлено вместо x. Вопреки сложившейся математической практике мы могли бы записать данное выражение и в таком виде: $<5(x)$. Здесь <5 — обозначение предиката "быть меньше 5". Данный предикат определяет множество всех действительных чисел, которые меньше числа 5. Обратите внимание, что мы некоторым конечным образом определили бесконечное множество.

Аналогично мы можем определить с помощью предиката $Красный(x)$ множество всех красных элементов. Разумеется, мы должны иметь алгоритм вычисления данного предиката, т. е. алгоритм определения, является ли предъявленный элемент x красным или нет.

В явном виде (экстенсионально) пустое множество обозначается как { }. Интенсионально пустое множество определяется через некий предикат, который ложен для всех элементов универсума.

Примечание

Предикаты в математической логике являются двузначными функциями. Однако как в математике, так и на практике нередко встречаются ситуации, к которым больше подходит многозначная и, в частности, трехзначная логика. Так, для некоторого элемента *x* результатом вычисления значений некоторого предиката *P*(*x*) может быть не только ИСТИНА или ЛОЖЬ, но и некоторое третье значение, например, `NULL`. Последнее интерпретируется как "не определено" или "не известно". Такая логика в том или ином виде принята при поддержке современных баз данных. Например, столбец таблицы базы данных может содержать помимо наборов каких-то символов особое значение `NULL`, которое понимается как "еще не введенное", "пока неизвестное" и т. п. Однако замечу, что понимание трехзначной логики лучше усваивается при понимании классической двузначной логики.

Между множествами определяется *отношение включения*. Так, множество A включается в множество B (это утверждение записывается как $A \subseteq B$), если каждый элемент множества A принадлежит и множеству B. В этом случае говорят, что множество A является подмножеством множества B. Два множества равны ($A = B$), если $A \subseteq B$ и $B \subseteq A$, т. е. оба множества включаются друг в друга и, следовательно, состоят из одних и тех же элементов. Очевидно, что любое множество является своим подмножеством, т. е. для любого множества A выполняется отношение $A \subseteq A$. Однако в общем случае из того, что $A \subseteq B$, еще не следует, что $B \subseteq A$.

Пустое множество $\varnothing$ включается в любое множество, т. е. для любого множества A имеет место включение $\varnothing \subseteq A$. Этот факт, хоть он и очевиден, можно доказать. Предположим, что $\varnothing \subseteq A$ ложно. Но это может быть только в том случае, когда существует некий элемент *x*, такой, что $x \in \varnothing$ и $x \notin A$. Однако это не возможно, т. к. пустое множество $\varnothing$ не имеет элементов по определению.

Не следует смешивать отношение принадлежности $\in$ между элементами и множествами и отношение включения $\subseteq$ между множествами. Так, например, если $x \in A$ и $A \in B$, то это еще не означает, что $x \in B$. Другой пример: если $A = \{B, a, c\}$ и $B = \{a, d\}$,

то $B \in A$, но не верно, что $B \subseteq A$. Верно лишь, что $\{B\} \subseteq A$, т. е. множество, состоящее из элемента *B*, является подмножеством множества *A*. Иначе говоря, элементы некоторого множества *A* сами могут быть множествами, но элементы последних не обязательно являются элементами множества *A*. Так, в рассматриваемом примере $a \in A$, $a \in B$, $d \in B$, но $d \notin A$. Заметим также, что элемент и множество, содержащее единственный этот элемент (т. е. x и $\{x\}$), — разные понятия.

Над множествами можно выполнять различные операции: объединение ($\cup$), пересечение ($\cap$), вычитание (–) и декартово произведение ($\times$).

Объединение множеств A и *B* есть множество, обозначаемое как $A \cup B$, которое содержит все элементы множества *A* и все элементы множества *B*. Пусть, например, $A = \{a, b, c, x, y\}$, $B = \{a, b, x, y, z\}$. Тогда $A \cup B = \{a, b, c, x, y, z\}$. Напомню, что в любом множестве все элементы должны быть различны. Другими словами, множество $A \cup B$ содержит только такие элементы, которые принадлежат множеству *A* или множеству *B* (возможно, и обоим одновременно). В частности, $A \cup \varnothing = A$.

Пересечение множеств A и *B* есть множество, обозначаемое как $A \cap B$, которое содержит все элементы, принадлежащие как множеству *A*, так и множеству *B*. Если таких элементов нет, то пересечение множеств пусто ($A \cap B = \varnothing$). Например, пусть $A = \{a, b, c, x, y\}$, $B = \{a, b, x, y, z\}$. Тогда $A \cap B = \{a, b, x, y\}$. В частности, $A \cap \varnothing = \varnothing$.

Вычитание из множества *A* множества *B* дает в результате множество, называемое *разностью* этих множеств, и обозначается как $A - B$. Это множество содержит все элементы множества *A*, которые не принадлежат множеству *B*. Например, пусть $A = \{a, b, c, d, x, y\}$, $B = \{a, b, x, y, z\}$. Тогда $A - B = \{c, d\}$. В частности, $A - \varnothing = A$. Если множества *A* и *B* не пересекаются (т. е. $A \cap B = \varnothing$), то $A - B = A$. Множество $A - B$ называют также *дополнением* множества *B* до множества *A*.