Глава 7

Определение значений

В этой главе...

- Использование переменных для уменьшения избыточного кодирования
- Получение часто запрашиваемой информации, находящейся в поле таблицы базы данных
- Комбинирование простых значений для создания составных выражений

В этой книге постоянно подчеркивается, насколько важной для поддержания целостности базы данных является структура этой базы. Впрочем, хотя значение структуры базы данных часто недооценивается, но не надо забывать, что наибольшую важность все же представляют сами данные. В конце концов, значения, хранящиеся на пересечении строк и столбцов в таблице базы данных, являются "сырьем", из которого можно получать ценную информацию об имеющихся взаимоотношениях и тенденциях.

Значения можно получать несколькими способами — непосредственно или с помощью функций или выражений. В этой главе описываются разные виды значений, функций и выражений.

Функция принимает данные и на их основе вычисляет значение. *Выражение* является комбинацией элементов данных, из которой SQL в результате вычислений получает единственное значение.

Значения

В SQL имеется несколько видов значений:

- ✓ значения типа записи;
- ✓ литеральные значения;
- ✓ переменные;
- ✓ специальные переменные;
- ✓ ссылки к столбцам.

Атомы не являются неделимыми

В девятнадцатом веке ученые верили, что атом является той минимальной частью материи, какая только возможна. Поэтому они и назвали эту часть *атомом* — словом, происходящим от греческого "атомос", что означает "неделимый". А теперь ученым известно, что атомы не являются неделимыми и состоят из протонов, нейтронов и электронов. Протоны и нейтроны, в свою очередь, состоят из кварков, глюонов и виртуальных кварков. Кто знает, может быть, и их нельзя назвать неделимыми?

Значение поля таблицы базы данных называется *атомарным*, хотя многие поля совсем не являются неделимыми. У значения типа `DATE` имеются следующие компоненты: месяц, год и день.

А компонентами значения типа TIMESTAMP являются час, минута, секунда и т.д. Значения типов REAL и FLOAT в качестве компонентов имеют экспоненту и мантиссу. В значении типа CHAR есть компоненты, к которым можно получить доступ с помощью SUBSTRING. Поэтому, по аналогии с атомами материи, название *"атомарные"* для значений полей баз данных все-таки правильно. Впрочем, если исходить из первоначального значения этого слова, то ни одно из современных применений термина *"атомарный"* правильным не является.

Значения типа записи

Самыми заметными значениями в базе данных являются табличные *значения типа записи*. Это значения, которые являются содержимым каждой строки, находящейся в таблице базы данных. Значение этого типа обычно состоит из множества компонентов, ведь в каждом столбце каждой строки всегда находится какое-либо значение. *Поле* — это пересечение столбца и строки. В поле содержится *скалярное*, или *атомарное*, значение. У этого значения имеется только один компонент.

Литеральные значения

В SQL *значение* может быть представлено или переменной, или константой. Было бы логично считать, что значение *переменной* время от времени меняется, а значение *константы* (т.е. постоянной величины) не меняется никогда. Важной разновидностью констант является *литеральное значение*. *Литерал* можно считать *WYSIWYG*-значением, потому что "то, что вы видите, то вы и получаете" (*W*hat *Y*ou *S*ee *I*s *W*hat *Y*ou *G*et). Представление литерального значения как раз и является этим самым значением.

Так как в языке SQL имеется много разных типов данных, то в нем имеется и много разных типов литералов. Некоторые примеры литералов разных типов данных приведены в табл. 7.1.

Обратите внимание, что литералы нечисловых типов заключены в апострофы. Эти знаки помогают избежать путаницы, хотя, впрочем, могут и привести к трудностям.

Таблица 7.1. Примеры литералов различных типов данных

Тип данных	Пример литерала
BIGINT	8589934592
INTEGER	186282
SMALLINT	186
NUMERIC	186282,42
DECIMAL	186282,42
REAL	6,02257E23
DOUBLE PRECISION	3,1415926535897E00
FLOAT	6,02257E23
CHARACTER(15)	'GREECE '
Примечание: в предыдущей строке в одинарные кавычки заключено пятнадцать символов и пробелов	
VARCHAR (CHARACTER VARYING)	'lepton'
NATIONAL CHARACTER(15)	'ΕΛΛΑΣ '[1]

[1] *Этот термин является словом, которым греки называют Грецию на своем языке. (Если написать его по-английски, то получится "Hellas", а по-русски — "Эллада".)*

Окончание табл. 7.1

Тип данных	Пример литерала
Примечание: в предыдущей строке в одинарные кавычки заключено пятнадцать символов и пробелов	
NATIONAL CHARACTER VARYING	'λεπτον'[2]
CHARACTER LARGE OBJECT (CLOB)	Очень длинная символьная стока
BINARY LARGE OBJECT (BLOB)	Очень длинная строка, состоящая из нулей и единиц (0 и 1)
DATE	DATE '1969-07-20'
TIME(2)	TIME '13.41.32.50'
TIMESTAMP(0)	TIMESTAMP '1998-05-17-13.03.16.000000'
TIME WITH TIMEZONE(4)	TIME '13.41.32.5000-08.00'
TIMESTAMP WITH TIMEZONE(4)	TIMESTAMP '1998-05-17-13.03.16.0000+02.00'
INTERVAL DAY	INTERVAL '7' DAY

А если литерал является символьной строкой, содержащей символ одинарной кавычки? В таком случае вместо одного этого символа в литерале должны быть две одинарные кавычки подряд, чтобы показать, что кавычка является частью строки и не указывает на ее завершение. Таким образом, чтобы получился символьный литерал 'Earth's atmosphere', необходимо ввести 'Earth''s atmosphere'.

Переменные

Прекрасно, когда при работе с базами данных можно манипулировать литералами и другими константами. Однако полезно иметь и переменные. Во многих случаях, не имея переменных, приходится делать намного больше работы. *Переменная* — это такая величина, значение которой может изменяться. Чтобы увидеть, почему переменные так полезны, рассмотрим следующий пример.

Предположим, что вы розничный продавец, у которого есть покупатели нескольких категорий. Тем из них, кто покупает товары в больших объемах, вы продаете эти товары по самым низким ценам. Тем же, кто покупает в средних объемах, вы продаете товары по ценам более высокого порядка. И наконец, те, кто ограничивается при покупках малыми объемами товаров, платят самую высокую цену. Вы хотите, чтобы все розничные цены имели определенные коэффициенты по отношению к той стоимости, в какую товары обошлись вам. Для своего товара F-117A вы решили, что покупатели товаров в больших объемах (покупатели класса C) будут за него платить в 1,4 раза больше, чем платите за этот товар вы. А покупатели товаров в средних объемах (покупатель класса B) будут уже платить в 1,5 раза больше. И наконец, покупатели товаров в малых объемах (покупатели класса A) — в 1,6 раза больше.

Вы храните значения стоимости товаров и назначаемых вами цен в таблице, которую вы назвали PRICING (ценообразование). Среди ее полей имеются такие: PRICE (цена), COST (стоимость), PRODUCT (продукт) и CLASS (класс). Чтобы реализовать свою новую структуру ценообразования, вы отправляете на выполнение следующие команды языка SQL:

```
UPDATE PRICING
   SET Price = Cost * 1.4
   WHERE Product = 'F-117A'
      AND Class = 'C' ;
```

[2] *Этот термин является словом "lepton" (лептон), написанным буквами греческого алфавита.*

```
UPDATE PRICING
   SET Price = Cost * 1.5
   WHERE Product = 'F-117A'
      AND Class = 'B' ;
UPDATE PRICING
   SET Price = Cost * 1.6
   WHERE Product = 'F-117A'
      AND Class = 'A' ;
```

Этот код прекрасный и пока что подходит для ваших нужд. А что если энергичные усилия конкурентов начинают подрывать ваш сектор рынка? Чтобы остаться на плаву, вам, возможно, придется уменьшить установленные вами значения разницы в ценах. Тогда потребуется ввести нечто похожее на строки следующих команд:

```
UPDATE PRICING
   SET Price = Cost * 1.25
   WHERE Product = 'F-117A'
      AND Class = 'C' ;
UPDATE PRICING
   SET Price = Cost * 1.35
   WHERE Product = 'F-117A'
      AND Class = 'B' ;
UPDATE PRICING
   SET Price = Cost * 1.45
   WHERE Product = 'F-117A'
      AND Class = 'A' ;
```

Если ваш рынок изменчив, то вам придется время от времени переписывать свой SQL-код. Это может потребовать значительных усилий с вашей стороны, особенно если цены указаны во многих местах вашего кода. Эти усилия можно свести к минимуму, если заменить литералы (например, 1.45) переменными (такими, например, как :multiplierA). Тогда свои операции обновления вы можете выполнять таким образом:

```
UPDATE PRICING
   SET Price = Cost * :multiplierC
   WHERE Product = 'F-117A'
      AND Class = 'C' ;
UPDATE PRICING
   SET Price = Cost * :multiplierB
   WHERE Product = 'F-117A'
      AND Class = 'B' ;
UPDATE PRICING
   SET Price = Cost * :multiplierA
   WHERE Product = 'F-117A'
      AND Class = 'A' ;
```

Теперь в любом случае, когда условия на рынке заставят вас менять ценообразование, остается только изменить значения переменных: :multiplierC, :multiplierB и :multiplierA. Эти переменные являются параметрами, передаваемыми SQL-коду, который затем использует полученные переменные, чтобы считать новые цены.

Иногда переменные, используемые таким образом, называются *параметрами*, а иногда — *базовыми переменными*. Переменные называются *параметрами*, если они находятся в приложениях, написанных на модульном языке SQL, а *базовыми переменными* — если используются во встроенном SQL.

Встроенный SQL означает, что операторы SQL встроены в код приложения, написанного на процедурном базовом языке. Кроме того, SQL-код можно поместить в модуль SQL. Модуль вызывается приложением, написанным на базовом языке. Каждый из этих двух методов имеет собственные преимущества и недостатки. Какой из них выбрать — это зависит от используемой вами конкретной реализации SQL.

Специальные переменные

Как только пользователь на клиентской машине соединяется с базой данных, находящейся на сервере, устанавливается *сеанс*. Если пользователь соединяется с несколькими базами данных, то сеанс, связанный с самым последним соединением, называется *текущим*, а *предыдущие сеансы* считаются *бездействующими*. Стандарт SQL:2003 определяет несколько специальных переменных, применяемых в многопользовательских системах. Эти переменные содержат данные о различных пользователях. Например, специальная переменная SESSION_USER (пользователь сеанса) содержит значение пользовательского идентификатора авторизации для текущего сеанса SQL. Вы можете написать программу мониторинга, определяющую, кто отправляет на выполнение операторы SQL, с помощью переменной SESSION_USER.

У модуля SQL может быть связанный с ним идентификатор авторизации, который определяется пользователем. Его значение хранится в переменной CURRENT_USER (текущий пользователь). Если такого идентификатора у модуля нет, то переменная CURRENT_USER имеет такое же значение, что и SESSION_USER.

В переменной SYSTEM_USER (системный пользователь) хранится идентификатор пользователя операционной системы. Он может отличаться от идентификатора этого пользователя, хранящегося в модуле SQL. Например, пользователь может регистрироваться в системе как LARRY (Ларри), а в модуле — уже как PLANT_MGR (директор завода). Таким образом, в переменной SESSION_USER будет храниться значение PLANT_MGR. Если этот пользователь явно не указывает идентификатор модуля, то значение PLANT_MGR будет храниться и в переменной CURRENT_USER. А значение LARRY будет храниться в переменной SYSTEM_USER.

Специальные переменные SYSTEM_USER, SESSION_USER и CURRENT_USER применяются для сбора данных о том, какие именно пользователи работают в системе. Вы можете поддерживать таблицу-журнал и периодически вставлять в нее значения, содержащиеся в этих переменных. Как это сделать, показано в следующем примере:

```
INSERT INTO USAGELOG (SNAPSHOT)
   VALUES ('User' SYSTEM_USER ||
      ' with ID ' || SESSION_USER ||
      ' active at ' || CURRENT_TIMESTAMP) ;
```

При выполнении этого оператора создаются примерно такие журнальные записи:

```
User LARRY with ID PLANT_MGR active at 1998-05-17-14:18:00
```

Ссылки к столбцам

В столбцах находятся значения, по одному в каждой табличной строке. Ссылки к таким значениям часто используются в операторах SQL. Полностью определенная ссылка к столбцу состоит из имени таблицы, точки и имени столбца (например, PRICING.Product). Посмотрите на следующий оператор:

```
SELECT PRICING.Cost
   FROM PRICING
   WHERE PRICING. Product = 'F-117A' ;
```

где PRICING. Product — это ссылка на столбец, которая содержит значение 'F-117A'. PRICING.Cost — это также ссылка на столбец, но вы не будете знать ее значения, пока не выполнится предшествующий этой ссылке оператор SELECT.

Так как имеет смысл делать ссылки только к тем столбцам, которые находятся в текущей таблице, то обычно эти ссылки полностью определять не нужно. Например, следующий оператор равнозначен предыдущему:

```
SELECT Cost
   FROM PRICING
   WHERE Product = 'F-117A' ;
```

Иногда все же приходится работать одновременно с разными таблицами. В базе данных у каких-либо двух таблиц могут быть столбцы с одинаковыми именами. В таком случае ссылки к этим столбцам приходится определять полностью. Это нужно для того, чтобы получаемый столбец был действительно тем, который вам нужен.

Предположим, например, что ваша компания имеет филиалы, расположенные в Кингстоне и Джефферсоне, и вы отдельно для каждого из этих филиалов ведете данные по работающим там сотрудникам. Ваша таблица по сотрудникам, работающим в Кингстоне, называется EMP_KINGSTON, а по работающим в Джефферсоне — EMP_JEFFERSON. Вам необходим список всех сотрудников, которые работают в обоих местах, поэтому следует найти всех тех, у кого имя вместе с фамилией находятся в обеих таблицах. То, что нужно, дает следующий оператор SELECT:

```
SELECT EMP_KINGSTON.FirstName, EMP_KINGSTON.LastName
   FROM EMP_KINGSTON, EMP_JEFFERSON
   WHERE EMP_KINGSTON.EmpID = EMP_JEFFERSON.EmpID ;
```

Так как идентификационный номер сотрудника является уникальным и имеет одно и то же значение независимо от филиала, в котором сотрудник работает, то этот номер можно использовать для связи между таблицами (в каждой из них он находится в столбце EmpID). В результате выполнения последнего оператора возвращаются имена и фамилии только тех сотрудников, чьи данные находятся в обеих таблицах. Имена и фамилии берутся соответственно из столбцов FirstName и LastName таблицы EMP_KINGSTON.

Выражения со значением

Выражение может быть простым или очень сложным. В нем могут находиться литеральные значения, имена столбцов, параметры, базовые переменные, подзапросы, логические связки и арифметические операторы. Впрочем, каким бы сложным выражение ни было, оно обязательно должно сводиться к одиночному значению.

Поэтому выражения SQL обычно называются *выражениями со значением*. Комбинирование множества таких выражений в одно возможно тогда, когда эти выражения-компоненты сводятся к значениям, имеющим совместимые типы данных.

В SQL определяется пять разных типов выражений со значением:

- ✓ строковые;
- ✓ числовые;
- ✓ даты-времени;
- ✓ интервальные;
- ✓ условные.

Строковые выражения со значением

Самым простым *строковым выражением со значением* является одиночное строковое значение. В более сложных выражениях могут быть также ссылки на столбцы, итоговые функции, скалярные подзапросы, выражения с использованием ключевых слов CASE и CAST или составные строковые выражения со значением. О выражениях со значением, использующих CASE и CAST, рассказывается в главе 8. В строковых выражениях со значением можно применять только один оператор — *оператор конкатенации*. Его можно применять к любым выражениям, чтобы, соединив их вместе, получить более сложное строковое выражение со значением. Оператор конкатенации представлен двумя вертикальными линиями (||). Некоторые примеры строковых выражений со значением показаны в следующей таблице.

Выражение	Результат
`'Хрустящий ' || 'арахис'`	`'Хрустящий арахис'`
`'Шарики' || ' ' || 'из желе'`	`'Шарики из желе'`
`FIRST_NAME || ' ' || LAST_NAME`	`'Джо Смит'`
`B'1100111' || B'01010011'`	`B'110011101010011'`
`'' || 'Спаржа'`	`'Спаржа'`
`'Спаржа' || ''`	`'Спаржа'`
`'С' || '' || 'пар' || '' || 'ж' || '' || 'а'`	`'Спаржа'`

Как показывают примеры из таблицы, если объединять какую-либо строку со строкой нулевой длины, то результат будет таким же, как и первоначальная строка.

Числовые выражения со значением

В *числовых выражениях со значением* к числовым данным можно применять операторы сложения, вычитания, умножения и деления. Такое выражение обязательно должно сводиться к числовому значению. Компоненты числового выражения со значением могут иметь разные типы данных или все могут быть числовыми. Тип данных результата зависит от типов данных компонентов, из которых получается этот результат. В стандарте SQL:2003 нет жесткого определения, каким образом тип данных результата, получаемого при выполнении выражения, должен зависеть от исходных компонентов этого выражения. Это объясняется различиями аппаратных платформ. Поэтому, если вы используете смешанные типы данных, обращайтесь к документации по той платформе, на которой работаете.

Вот некоторые примеры числовых выражений со значением.

- ✓ -27
- ✓ 49 + 83
- ✓ 5 * (12 - 3)
- ✓ PROTEIN + FAT + CARBOHYDRATE
- ✓ FEET/5280
- ✓ COST * :multilierA

Выражения со значением даты-времени

Выражения со значением даты-времени выполняют операции с данными, относящимися к дате и времени. Компоненты этих выражений могут иметь типы данных DATE, TIME, TIMESTAMP и INTERVAL. Результат выполнения выражения со значением даты-времени все-

гда относится к одному из типов даты-времени (DATE, TIME или TIMESTAMP). Например, после выполнения следующего выражения будет получена дата, которая наступит ровно через неделю:

```
CURRENT_DATE + INTERVAL '7' DAY
```

Значения времени поддерживаются в координатах Всемирного времени (Universal Time Coordinates, UTC), ранее известных как время по Гринвичу. Однако можно указывать и смещение, чтобы время соответствовало текущему часовому поясу. Для местного часового пояса, применяемого в вашей системе, можно использовать простой синтаксис, пример которого приведен ниже.

```
TIME '22.55.00' AT LOCAL
```

Кроме того, это значение можно указать и более развернуто:

```
TIME '22.55.00' AT TIME ZONE INTERVAL '-08.00' HOUR TO MINUTE
```

Последнее выражение определяет местное время часового пояса, в котором находится город Портленд, штат Орегон. Этот часовой пояс отстоит от Гринвича на восемь часов.

Интервальные выражения со значением

Если взять два значения даты-времени и от одного из них отнять другое, то получится *интервал*. Но сложение таких значений друг с другом не имеет смысла, поэтому SQL эту операцию не поддерживает. Если же сложить друг с другом два интервала или вычесть один из другого, то в результате снова получится интервал. Кроме того, интервал можно умножать или делить на числовую константу.

Вспомните, что в SQL имеются два типа интервалов: *год-месяц* и *день-время*. Чтобы избежать двусмысленности, необходимо в интервальном выражении со значением указывать, какой из этих типов в нем используется. Например, в следующем выражении вычисляется интервал в годах и месяцах от текущей даты до дня, когда вы достигнете пенсионного возраста (60 лет):

```
(BIRTHDAY_60 - CURRENT_DATE) YEAR TO MONTH
```

А это возвращает интервал в 40 дней:

```
INTERVAL '17' DAY + INTERVAL '23' DAY
```

Ниже приблизительно подсчитывается общее число месяцев, в течение которых мать пятерых детей была беременна при условии, что сейчас она не ждет шестого.

```
INTERVAL '9' MONTH * 5
```

Интервалы могут быть как положительные, так и отрицательные и состоять из любого выражения со значением или комбинации таких выражений, значением которой является интервал.

Условные выражения со значением

Значение *условного выражения со значением* зависит от условия. Такие выражения, как CASE, NULLIF и COALESCE, значительно сложнее, чем другие выражения со значением. Эти три вида условных выражений настолько сложны, что заслуживают отдельного рассмотрения. Подробно о них речь пойдет в главе 8.

Функции

Функция — это простая или достаточно сложная операция, которую обычные команды SQL выполнить не могут, но которая тем не менее достаточно часто встречается на практике. В SQL имеются функции, выполняющие работу, которую иначе пришлось бы выполнять приложению, написанному на базовом языке. В языке SQL есть две главные разновидности функций: *итоговые функции* и *функции значения*.

Суммирование с помощью итоговых функций

Итоговые функции применяются к наборам строк из таблицы, а не только к ее отдельным строкам. Эти функции в текущем наборе строк "суммируют" некоторые характеристики, т.е. получают по ним определенные итоги. В такой набор могут входить все строки таблицы или только те из них, которые определяются предложением WHERE. (Подробно о предложениях WHERE рассказывается в главе 9.)

Программисты используют название *итоговые функции*, потому что те берут информацию из целого набора строк, определенным образом ее обрабатывают и выдают результат в виде единичной строки. Кроме того, эти функции еще называются *функциями наборов*.

Чтобы показать применение итоговых функций, проанализируйте табл. 7.2, в которой представлены питательные компоненты, содержащиеся в 100 граммах некоторых продуктов питания.

Таблица 7.2. Питательные компоненты некоторых продуктов питания (в 100 граммах)

Продукт питания (Food)	Калории (Calories)	Белки (Protein), г	Жиры (Fat), г	Углеводы (Carbogidrate), г
Жареные миндальные орехи	627	18,6	57,7	19,6
Спаржа	20	2,2	0,2	3,6
Сырые бананы	85	1,1	0,2	22,2
Гамбургер с нежирной говядиной	219	27,4	11,3	
Нежное мясо цыплят	166	31,6	3,4	
Жареный опоссум	221	30,2	10,2	
Свиной окорок	394	21,9	33,3	
Фасоль лима	111	7,6	0,5	19,8
Кола	39			10,0
Белый хлеб	269	8,7	3,2	50,4
Пшеничный хлеб	243	10,5	3,0	47,7
Брокколи	26	3,1	0,3	4,5
Сливочное масло	716	0,6	81,0	0,4
Шарики из желе	367		0,5	93,1
Хрустящий арахис	421	5,7	10,4	81,0

Информация из табл. 7.2 хранится в таблице FOODS (продукты), находящейся в базе данных. В пустых полях находится значение NULL. Сообщить важные сведения о данных из этой таблицы помогают итоговые функции COUNT, AVG, MAX, MIN и SUM.

COUNT

Функция COUNT (счет) сообщает, сколько строк находится в таблице или сколько строк таблицы удовлетворяют некоторые условия. Вот самое простое применение этой функции:

```
SELECT COUNT (*)
   FROM FOODS ;
```

Функция возвращает результат, равный 15, так как она считает все строки таблицы FOODS. Тот же результат получается при выполнении следующего оператора:

```
SELECT COUNT (CALORIES)
   FROM FOODS ;
```

Так как значение в столбец CALORIES (калории) было введено в каждой строке, то результат подсчета получается тот же. Правда, если в некоторых строках в этом столбце находятся неопределенные значения, функция такие строки не считает.

Следующий оператор возвращает значение, равное 11, так как в 4 из 15 строк таблицы FOODS в столбце CARBOHYDRATE (углеводы) находится значение NULL:

```
SELECT COUNT (CARBOHYDRATE)
   FROM FOODS ;
```

В поле таблицы базы данных неопределенное значение может находиться по разным причинам. Самыми распространенными являются следующие: значение вообще не известно или пока не известно. Или значение, возможно, известно, но пока еще не введено. Иногда, если значение, предназначенное для какого-либо поля, равно нулю, оператор, вводящий данные, это поле обходит стороной и, таким образом, оставляет в нем значение NULL. Так поступать не надо, потому что нуль все же является определенным значением и его можно учитывать при подсчетах. А NULL определенным значением *не является*, и в SQL неопределенные значения не учитываются при расчетах.

Кроме того, чтобы узнать, сколько в столбце имеется различных значений, можно использовать функцию COUNT в сочетании с ключевым словом DISTINCT (различный). Проанализируйте следующий оператор:

```
SELECT COUNT (DISTINCT Fat)
   FROM FOODS ;
```

Он возвращает значение, равное 12. Как видно в таблице, в 100-граммовой порции спаржи имеется столько жиров (0,2 грамма), сколько и в 100 граммах бананов, а в 100-граммовой порции фасоли лима — ровно столько жиров (0,5 грамма), сколько в 100 граммах желе. Таким образом, в таблице находится всего 12 разных значений, имеющих отношение к содержанию жиров.

AVG

Функция AVG (среднее) подсчитывает и возвращает среднее арифметическое всех значений, находящихся в определенном столбце. Конечно, эту функцию можно применять только к столбцам с числовыми данными, как в следующем примере:

```
SELECT AVG (Fat)
   FROM FOODS ;
```

В результате получается среднее содержание жиров, равное 15,37. Это число достаточно высокое. Дело в том, что весь подсчет портит информация по сливочному маслу. Возможно, вы зададите себе вопрос, а каким было бы среднее содержание жиров, если бы не учитывалось масло. Чтобы ответить на него, в оператор можно поместить выражение WHERE:

```
SELECT AVG (Fat)
   FROM FOODS
   WHERE FOOD <> 'Butter' ;
```

В этом случае содержание жиров в 100 граммах пищевых продуктов в среднем падает до 10,32 грамма.

MAX

Функция MAX (максимум) возвращает максимальное значение, найденное в указанном столбце. Следующий оператор возвращает значение, равное 81 (количество жиров в 100 граммах сливочного масла):

```
SELECT MAX (Fat)
   FROM FOODS ;
```

MIN

Функция MIN (минимум) возвращает минимальное значение, обнаруженное в указанном столбце. Следующий оператор возвращает значение, равное 0,4, потому что функция не учитывает неопределенные значения:

```
SELECT MIN (Carbohydrate)
   FROM FOODS ;
```

SUM

Функция SUM (сумма) возвращает сумму всех значений, обнаруженных в указанном столбце. Следующий оператор возвращает число 3924, которое является общим количеством калорий во всех 15 продуктах:

```
SELECT SUM (Calories)
   FROM FOODS ;
```

Функции значения

Некоторые операции применяются для самых разных целей. Так как эти операции приходится использовать достаточно часто, то было бы более чем оправданным включить их в SQL в виде функций значения. Конечно, если сравнивать с такими системами управления базами данных, как Access или dBASE, то в SQL этих функций довольно-таки мало, но те немногие, что есть, являются, вероятно, функциями, которые нужны вам чаще всего. В SQL используются три вида таких функций:

- ✓ строковые функции;
- ✓ числовые функции;
- ✓ функции даты-времени.

Строковые функции

Строковые функции принимают значение одной символьной или битовой строки и возвращают другую символьную или битовую строку. В SQL имеется шесть таких функций:

- ✓ SUBSTRING;
- ✓ UPPER;
- ✓ LOWER;
- ✓ TRIM;
- ✓ TRANSLATE;
- ✓ CONVERT.

SUBSTRING

Функция SUBSTRING (подстрока) используется для того, чтобы из исходной строки выделить подстроку. Выделенная функцией подстрока имеет тот же тип, что и исходная. Например, если исходная строка является символьной, то и подстрока также является символьной. Вот синтаксис функции SUBSTRING:

```
SUBSTRING (строковое_значение FROM начало [FOR длина]).
```

Предложение в квадратных скобках ([]) не является обязательным. Подстрока, которую следует выделить из *строкового_значения*, начинается с символа, порядковый номер которого, если считать с самого первого символа, представлен значением *начало*. Кроме того, подстрока состоит из определенного количества символов, представленного значением *длина*. Если

предложение FOR отсутствует, то подстрока выделяется, начиная от символа, соответствующего значению *начало*, до самого конца строки. Проанализируйте следующий пример:

```
SUBSTRING ('Полностью пшеничный хлеб' FROM 11 FOR 11)
```

Выделенной подстрокой является 'пшеничный х'. Она начинается с одиннадцатого символа исходной строки и имеет длину в одиннадцать символов. С первого взгляда SUBSTRING не представляется такой уж ценной функцией. Для литерала 'Полностью пшеничный хлеб' не требуется функция нахождения подстроки. Впрочем, функция SUBSTRING действительно представляет ценность, потому что строковое значение не обязательно должно быть литералом. Это значение может быть любым выражением, в результате выполнения которого получается символьная строка. Например, это может быть переменная fooditem, которая каждый раз может принимать разные значения. Следующее выражение может извлекать нужную подстроку независимо от того, какую символьную строку представляет переменная fooditem:

```
SUBSTRING (:fooditem FROM 11 FOR 11).
```

Все функции значения объединяет то, что они могут оперировать как со значениями, так и с выражениями, после выполнения которых получаются значения требуемого типа.

При использовании функции SUBSTRING не следует забывать о следующем. Выбираемая вами подстрока действительно должна быть частью исходной строки. Если вам нужна подстрока, которая начинается с одиннадцатого символа, а в исходной строке всего только четыре символа, то вы получите значение NULL. Поэтому необходимо иметь некоторое представление о структуре своих данных, перед тем как задавать значения для функции SUBSTRING. Кроме того, нельзя указывать отрицательную длину подстроки.

Если у столбца тип данных VARCHAR, ширина поля конкретной строки не будет известна. Если вы укажете слишком большую длину подстроки, при которой та выйдет за правый край поля, функция SUBSTRING возвратит конец исходной строки и не будет сообщать об ошибке.

Скажем, у вас имеется оператор

```
SELECT * FROM FOODS
   WHERE SUBSTRING (Food FROM 7 FOR 7) = 'хлеб' ;
```

И даже если значение, находящееся в столбце FOOD таблицы FOODS, имеет длину меньше 14 символов, этот оператор все равно возвращает табличную строку с данными, относящимися к белому хлебу.

Если какой-либо операнд функции SUBSTRING имеет значение NULL, то эта функция возвращает результат NULL.

UPPER

Другая функция, UPPER (верхний регистр), преобразует все символы символьной строки в верхний регистр, как показано в следующей таблице, в примерах со строками 'e.e.cummings' и 'Isaac Newton, Ph.D.'.

Выражение	Результат
`UPPER ('e.e.cummings')`	`'E.E.CUMMINGS'`
`UPPER ('Isaac Newton, Ph.D.')`	`'ISAAC NEWTON, PH.D.'`

Функция UPPER не оказывает воздействия на строку, все символы которой уже находятся в верхнем регистре.

LOWER

Другая функция, LOWER (нижний регистр), преобразует все символы символьной строки в нижний регистр, как показано в следующей таблице, в примерах со строками 'TAXES' и 'E.E.Cummings'.

Выражение	Результат
`LOWER ('TAXES')`	`'taxes'`
`LOWER ('E. E. Cummings')`	`'e. e. cummings'`

Функция LOWER не оказывает воздействия на строку, все символы которой уже находятся в нижнем регистре.

TRIM

Чтобы из символьной строки удалить ведущие, замыкающие или одновременно и те и другие пробелы (и не только пробелы), используйте функцию TRIM (обрезать). Следующие примеры показывают, как ее использовать, например, применительно к строкам, где находится слово treat.

Выражение	Результат
`TRIM (LEADING ' ' FROM ' treat ')`	`'treat '`
`TRIM (TRAILING ' ' FROM ' treat ')`	`' treat'`
`TRIM (BOTH ' ' FROM ' treat ')`	`'treat'`
`TRIM (BOTH 't' FROM 'treat')`	`'rea'`

Символом по умолчанию для этой функции является пробел, поэтому следующий синтаксис также правильный:

```
TRIM (BOTH FROM ' treat ').
```

В этом случае получается тот же результат, что и в третьем примере из таблицы, а именно 'treat'.

TRANSLATE и CONVERT

Функции TRANSLATE (перевести) и CONVERT (преобразовать) выбирают исходную строку, составленную из символов одного набора, и переводят ее в строку, составленную из символов другого набора. Примерами могут быть переводы символов из английского набора в армянский или символов иврита во французский. Функции преобразования, выполняющие эти действия, зависят от реализации SQL. Подробности можно узнать в документации по имеющейся у вас реализации.

Если бы перевод с одного языка на другой был таким легким, как вызов в SQL функции TRANSLATE, то это было бы прекрасно. К сожалению, такая задача — не из легких. Все, что осуществляет TRANSLATE, — это перевод символа из первого символьного набора в соответствующий символ из второго набора. Она может, например, перевести ‘ELLAS’ (Греция) в 'Ellas'. Однако функция TRANSLATE не может перевести ‘ELLAS’ в 'Greece' (Греция).

Числовые функции

Числовые функции значения могут принимать данные разных типов, но возвращают всегда числовое значение. В SQL имеется тринадцать таких функций.

- ✓ Положение (POSITION).
- ✓ Извлечение (EXTRACT).
- ✓ Длина (CHAR_LENGHT, CHARACTER_LENGHT, OCTET_LENGHT).
- ✓ Кардинальные числа (CARDINALITY).
- ✓ Абсолютное значение (ABS).
- ✓ Остаток от деления нацело (MOD).
- ✓ Натуральный логарифм (LN).
- ✓ Экспонента (EXP).
- ✓ Возведение в степень (POWER).
- ✓ Квадратный корень (SQRT).
- ✓ Округление "вниз" (FLOOR).
- ✓ Округление "вверх" (CEIL, CEILING).
- ✓ Интервальный номер (WIDTH_BUCKET).

POSITION

Функция POSITION (положение) ищет указанную целевую строку внутри указанной исходной и возвращает положение в ней начального символа целевой строки. Эта функция имеет такой синтаксис:

```
POSITION (целевая_строка IN исходная_строка)
```

В следующей таблице приведено несколько примеров использования POSITION для исходной строки 'Полностью пшеничный хлеб'.

Выражение	Результат
`POSITION ('П' IN 'Полностью пшеничный хлеб')`	`1`
`POSITION ('Пол' IN 'Полностью пшеничный хлеб')`	`1`
`POSITION ('пш' IN 'Полностью пшеничный хлеб')`	`11`
`POSITION ('пшо' IN 'Полностью пшеничный хлеб')`	`0`
`POSITION ('' IN 'Полностью пшеничный хлеб')`	`1`

Если эта функция не находит целевую строку, то возвращает неопределенное значение. А если у целевой строки нулевая длина (как в последнем примере), то функция POSITION всегда возвращает единицу. Если любой из операндов этой функции имеет значение NULL, то в результате ее выполнения получится NULL.

EXTRACT

Функция EXTRACT (извлечь) извлекает одиночное поле из значения типа даты-времени или интервала. Например, следующее выражение возвращает 08:

```
EXTRACT (MONTH FROM DATE '2000-08-20').
```

CHARACTER_LENGTH

Функция CHARACTER_LENGTH (длина в символах) возвращает количество символов, находящихся в символьной строке. Например, следующее выражение возвращает 15:

```
CHARACTER_LENGTH ('Жареный опоссум')
```

То, что уже говорилось в этой главе по поводу функции SUBSTRING, относится и к CHARACTER_LENGTH — эта функция не особенно полезна, если ее аргументами являются литералы, например, такие как 'Жареный опоссум'. Вместо выражения CHARACTER_LENGTH ('Жареный опоссум') можно написать число 15. Действительно, написать '15' проще. Функция SUBSTRING становится более полезной, если ее аргумент является не литеральным значением, а выражением.

OCTET_LENGTH

Что касается мира музыки, то в нем вокальный ансамбль, состоящий из восьми певцов, называется *октетом*. В этом ансамбле обычно есть первое и второе сопрано, первый и второй альт, первый и второй тенор, а также первый и второй бас. А что касается мира информатики, то в нем ансамбль, состоящий из восьми битов данных, называется *байтом*. Слово *байт* ясно показывает, что оно имеет отношение к *биту*, но при этом подразумевается нечто большее. Прекрасная игра слов, но, к сожалению, в слове “байт” ничто не напоминает о “восьмеричности”. А если позаимствовать музыкальный термин, то тогда набор из восьми битов будет иметь более подходящее и более описательное название.

Практически во всех современных компьютерах для представления одного алфавитно-цифрового символа используется восемь битов. А в более сложных наборах символов (таких, например, как китайский) для этого требуется уже 16 битов. Функция OCTET_LENGTH (длина в октетах) подсчитывает и возвращает количество октетов (байтов), находящихся в строке. Если строка является битовой, то эта функция возвращает такое количество октетов, чтобы вместить находящееся в строке количество битов. А если строка состоит из символов англоязычного набора (с одним октетом на символ), то OCTET_LENGTH возвращает количество символов, имеющихся в строке. Если же строка состоит из символов китайского набора, то тогда число, возвращаемое функцией, в два раза превышает количество китайских символов. Например:

```
OCTET_LENGTH ('Beans, Lima')
```

Эта функция возвращает 11, потому что каждый символ помещается в октете.

В некоторых наборах символов для разных символов используется разное количество октетов. В частности, в тех из них, которые поддерживают смешанное использование символов канджи и латиницы, для перехода из одного набора символов в другой используются *управляющие последовательности*. Например, для строки, состоящий из 30 латинских символов, потребуется 30 октетов. А если все ее 30 символов взяты из канджи, то для этой строки нужно 62 октета (60 октетов плюс ведущий и замыкающий символы переключения). И наконец, если в этой строке символы латиницы и канджи попеременно чередуются друг с другом, то тогда для нее требуется 150 октетов. (Тогда для каждого символа канджи требуется два октета, а также по одному октету для ведущего и замыкающего символа переключения.) Функция OCTET_LENGTH возвращает то количество октетов, которое нужно, чтобы поместить в них имеющуюся строку.

CARDINALITY

Эта функция работает с коллекциями элементов, такими как массивы или мультимножества, где каждый элемент является значением определенного типа данных. Кардинальное число коллекции — это количество содержащихся в ней элементов. Рассмотрим один из примеров использования функции CARDINALITY:

```
CARDINALITY (TeamRoster)
```

Например, если в списке членов команды значится двенадцать человек, то эта функция возвращает значение 12. Столбец TeamRoster таблицы TEAM может быть как массивом, так и мультимножеством. В свою очередь, массив — это упорядоченная коллекция элементов, а мультимножество — неупорядоченная коллекция. Для списка команды, который может изменяться достаточно часто, разумней использовать мультимножество.

ABS

Функция ABS возвращает абсолютное значение числового выражения.

```
ABS (-273)
```

Функция возвращает 273.

MOD

Функция MOD возвращает остаток от деления нацело первого числового выражения на второе числовое выражение.

```
MOD (3,2)
```

Данная функция возвращает числовое значение 1 — остаток, получаемый при делении нацело числа 3 на число 2.

LN

Функция LN возвращает натуральный логарифм от числового.

```
LN (9)
```

Возвращаемое значение будет приближенно равно 2,197224577. Количество знаков числа после запятой зависит от вашей реализации.

EXP

Данная функция возводит основание натурального логарифма *e* в степень, указанную числовым выражением.

```
EXP (2)
```

Функция возвращает значение, которое приближенно равно 7,389056. Количество знаков числа после запятой зависит от вашей реализации.

POWER

Функция POWER возводит первое числовое выражение в степень, указанную вторым числовым выражением.

```
POWER (2,8)
```

В этом примере функция возвращает значение 256, т.е. два в восьмой степени.

SQRT

Эта функция возвращает квадратный корень числового выражения.

```
SQRT (4)
```

Функция возвращает значение 2 — квадратный корень из четырех.

FLOOR

Функция FLOOR округляет числовое выражение до наибольшего целого числа, не превышающего данное выражение.

```
FLOOR (3,141592)
```

Функция возвращает значение 3,0.

CEIL, или CEILING

Данная функция округляет числовое выражение до наименьшего целого числа, которое не меньше, чем данное выражение.

```
CEIL (3,141592)
```

Функция возвращает значение 4,0.

WIDTH_BUCKET

Функция WIDTH_BUCKET используется при выполнении процессов в режиме реального времени (online application processing, OLAP). Эта функция имеет четыре аргумента и возвращает целое число между 0 (нулем) и значением последнего аргумента плюс 1 (один). Для первого аргумента она назначает область в разделенном на равновеликие части диапазоне чисел между вторым и третьим аргументами функции. Для значений, находящихся за пределами заданного диапазона, функция возвращает значение 0 (нуль) либо значение последнего аргумента плюс 1 (один).

Например:

```
WIDTH_BUCKET (PI, 0, 9, 5)
```

Предположим, что PI — числовое выражение со значением — это 3,141592. Интервал значений между нулем и девятью (0 и 9 — второй и третий аргументы функции соответственно) нужно разделить на пять равных отрезков (5 — четвертый аргумент функции), каждый шириной в две единицы. В этом случае функция возвращает значение 2, поскольку число 3,141592 находится во втором отрезке, который является диапазоном значений от двух до четырех.

Введение операторов языка SQL в базу данных приложения Microsoft Access

Access не позволяет ввести операторы SQL в базу данных, поэтому все операторы должны быть введены как запросы. Многие программные продукты, например SQL Server, Oracle, MySQL или PostgreSQL, имеют редакторы, предназначенные для ввода операторов языка SQL. Так, в приложении SQL Server это редактор Query Analyzer. Для других приложений такие редакторы описаны в документации.

Для Access также существует возможность ввода операторов SQL, однако этот путь весьма сложен и зачастую запутан. Подробные пошаговые инструкции относительно ввода операторов SQL в приложение Access см. в главе 4.

Функции значения даты-времени

В языке SQL имеются три функции, которые возвращают информацию о текущей дате, текущем времени или о том и другом вместе. CURRENT_DATE возвращает текущую дату, CURRENT_TIME — текущее время, а CURRENT_TIMESTAMP — текущую дату и текущее время. Первая из этих функций не принимает аргументов, а вторые две — только один. Этот аргумент указывает точность для секундной части возвращаемого функцией значения времени. О типах данных *даты-времени* и о том, что такое точность, см. в главе 2.

Некоторые примеры функций значения даты-времени приведены в следующей таблице.

Выражение	Результат
CURRENT_DATE	2000-12-31
CURRENT_TIME (1)	08:36:57.3
CURRENT_TIMESTAMP (2)	2000 12 31 08:36:57.38

Дата, возвращаемая функцией CURRENT_DATE, имеет тип данных не CHARACTER, а DATE. Время, возвращаемое функцией CURRENT_TIME (*p*), имеет, в свою очередь, тип данных TIME, значение даты и времени, возвращаемое функцией CURRENT_TIMESTAMP (*p*), имеет тип данных TIMESTAMP. Так как информацию о дате и времени средства SQL получают из системных часов компьютера, то эта информация является правильной для того часового пояса, в котором находится компьютер.

В некоторых приложениях значения даты и времени требуется представлять в виде символьных строк. Преобразование типа данных можно выполнять с помощью выражения CAST (приведение), которое описывается в главе 8.