8. Transact SQL как диалект языка SQL

Оглавление

8.1. Основные объекты SQL

Языком компонента Database Engine является Transact-SQL, который обладает основными свойствами любого другого распространенного языка программирования. Эта такие свойства, как:

- литералы (или константы);
- ограничители;
- комментарии;
- идентификаторы;
- зарезервированные ключевые слова.

8.1.1. Литералы

Литерал — это буквенно-цифровая (строковая), шестнадцатеричная или числовая константа. Строковая константа содержит один или больше символов определенного набора символов, заключенных между одинарными (' ') или двойными (" ") прямыми кавычками. (Константы предпочтительно заключать в одинарные кавычки, т. к. двойные кавычки используются во многих других случаях, как мы увидим вскоре.) Чтобы вставить одинарную кавычку в строку, заключенную в одинарные кавычки, нужно использовать две одинарные кавычки последовательно. Шестнадцатеричные константы используются для представления непечатаемых символов и прочих двоичных данных. Они начинаются символами 0x, за которыми следует четное число буквенных или цифровых символов.

К числовым константам относятся все целочисленные значения и значения с фиксированной и плавающей запятой (точкой).

Константы всегда обладают такими свойствами, как тип данных и длина, которые оба зависят от формата константы. Кроме этого, числовые константы имеют дополнительные свойства — точность и коэффициент масштабирования (scale factor).

8.1.2. Ограничители

В языке Transact-SQL двойные кавычки имеют два разных применения. Кроме применения для заключения строк, они также могут использоваться в качестве ограничителей для так называемых идентификаторов с ограничителями (delimited identifier). Идентификатор с ограничителями — это особый вид идентификатора, который обычно применяется для того, чтобы разрешить использование ключевых слов в качестве идентификаторов, а также разрешить использование пробелов в именах объектов баз данных.

Применение двойных кавычек в языке Transact-SQL определяется с помощью параметра QUOTED_IDENTIFIER инструкции SET. Если этому параметру присвоено значение ON (значение по умолчанию), идентификаторы, заключенные в двойные кавычки, будут определяться как идентификаторы с ограничителями. В таком случае двойные кавычки нельзя будет применять для ограничения строк.

8.1.3. Комментарии

В языке Transact-SQL существует два способа определения комментариев. В первом, текст, заключенный между парами символов /* и */, является комментарием. Такой комментарий может занимать несколько строк. В другом способе два символа дефиса (--) указывают, что следующий за ними до конца текущей строки текст является комментарием. (Способ обозначения комментариев двумя дефисами отвечает стандарту ANSI SQL, а способ с помощью символов /* и */ является расширением языка Transact-SQL.)

8.1.4. Идентификаторы

В языке Transact-SQL идентификаторы применяются для обозначения объектов баз данных, таких как собственно базы данных, их таблицы и индексы. Идентификатор состоит из строки длиной до 128 символов, которая может содержать буквы, цифры и символы _, @, # и &. Первым символом идентификатора должна быть буква или символ _, @ или #. Символ # в начале имени таблицы или хранимой процедуры обозначает временный объект, а символ @ обозначает переменную. Как упоминалось ранее, эти правила не относятся к идентификаторам с ограничителями (также называемыми идентификаторами в скобках), которые могут содержать или начинаться с любого символа, иного, чем сам ограничитель.

8.1.5. Зарезервированные ключевые слова

Каждый язык программирования имеет набор зарезервированных имен, которые требуется писать и применять в определенном формате. Такие имена называются зарезервированными ключевыми словами. В языке Transact-SQL используются разные виды таких имен, которые, как и во многих других языках программирования, нельзя применять в качестве имен объектов, за исключением, когда они используются для этой цели, как идентификаторы с ограничителями (или идентификаторы в скобках).

8.2. Типы данных

Все значения в столбце должны быть одного типа данных. (Единственным исключением из этого правила являются значения типа данных SQL_VARIANT.) Используемые в Transact-SQL типы данных можно разбить на следующие категории:

- числовые типы;
- символьные типы;
- временные типы (даты и/или времени);
- прочие типы данных.

8.2.1. Числовые типы данных

Таблица 8.1. Числовые типы данных

Тип данных	Описание
INTEGER	Представляет целочисленные значения длиной в 4 байта в диапазоне от –2 147 483 648 до 2 147 483 647. INT — сокращенная форма от INTEGER
SMALLINT	Представляет целочисленные значения длиной в 2 байта в диапазоне от –32 768 до 32 767
TINYINT	Представляет целочисленные значения длиной в 1 байт в диапазоне от 0 до 255
BIGINT	Представляет целочисленные значения длиной в 8 байт в диапазоне от -2^{63} до $2^{63} - 1$
DECIMAL(p,[s])	Представляет значения с фиксированной запятой (точкой). Аргумент p (precision — точность) указывает общее количество разрядов, а аргумент s (scale — степень) — количество разрядов справа от полагаемой десятичной точки. В зависимости от значения аргумента p, значения DECIMAL сохраняются в 5 до 17 байтах. DEC — сокращенная форма от DECIMAL
NUMERIC(p,[s])	Синоним DECIMAL
REAL	Применяется для представления значений с плавающей точкой. Диапазон положительных значений простирается приблизительно от 2,23E –308 до 1,79E +308, а отрицательных приблизительно от –1,18E –38 до –1,18E +38. Также может быть представлено и нулевое значение
FLOAT[(p)]	Подобно типу REAL, представляет значения с плавающей точкой. Аргумент p определяет точность. При значении p < 25 представляемые значения имеют одинарную точность (требуют 4 байта для хранения), а при значении p >= 25 — двойную точность (требуют 8 байтов для хранения)
MONEY	Используется для представления денежных значений. Значения типа MONEY соответствуют 8-байтовым значениям типа DECIMAL, округленным до четырех разрядов после десятичной точки
SMALLMONEY	Представляет такие же значения, что и тип MONEY, но длиной в 4 байта

8.2.2. Символьные типы данных

Существует два общих вида символьных типов данных. Строки могут представляться однобайтовыми символами или же символами в кодировке Unicode. (В кодировке Unicode для представления одного символа применяется несколько байтов.) Кроме этого, строки могут быть разной длины.

Таблица 8.2. Символьные типы данных

Тип данных	Описание
CHAR[(n)]	Применяется для представления строк фиксированной длины, состоящих из n однобайтовых символов. Максимальное значение n равно 8000. CHARACTER(n) — альтернативная эквивалентная форма CHAR(n). Если n явно не указано, то его значение полагается равным 1
VARCHAR[(n)]	Используется для представления строки однобайтовых символов переменной длины (0 < n < 8 000). В отличие от типа данных CHAR, количество байтов для хранения значений типа данных VARCHAR равно их действительной длине. Этот тип данных имеет два синонима: CHAR VARYING и CHARACTER VARYING
NCHAR[(n)]	Используется для хранения строк фиксированной длины, состоящих из символов в кодировке Unicode. Основная разница между типами данных CHAR и NCHAR состоит в том, что для хранения каждого символа строки типа NCHAR требуется 2 байта, а строки типа CHAR — 1 байт. Поэтому строка типа данных NCHAR может содержать самое большее 4000 символов

NVARCHAR[(n)]	Используется для хранения строк переменной длины, состоящих из символов в кодировке Unicode. Основная разница между типами данных VARCHAR и NVARCHAR состоит в том, что для хранения каждого символа строки типа NVARCHAR требуется 2 байта, а строки типа VARCHAR — 1 байт. Поэтому строка типа данных NVARCHAR может содержать самое большее 4000 символов

8.2.3. Временные типы данных

В языке Transact-SQL поддерживаются следующие временные типы данных:

- DATETIME;
- SMALLDATETIME;
- DATE;
- TIME;
- DATETIME2;
- DATETIMEOFFSET.

Типы данных DATETIME и SMALLDATETIME применяются для хранения даты и времени в виде целочисленных значений длиной в 4 и 2 байта соответственно. Значения типа DATETIME и SMALLDATETIME сохраняются внутренне как два отдельных числовых значения. Составляющая даты значений типа DATETIME хранится в диапазоне от 01/01/1753 до 31/12/9999, а соответствующая составляющая значений типа DATETIME — в диапазоне от 01/01/1900 до 06/06/2079. Составляющая времени хранится во втором 4-байтовом (2-байтовом для значений типа SMALLDATETIME) поле в виде числа трехсотых долей секунды (для DATETIME) или числа минут (для SMALLDATETIME), истекших после полуночи.

Если нужно сохранить только составляющую даты или времени, использование значений типа DATETIME или SMALLDATETIME несколько неудобно. По этой причине в SQL Server были введены типы данных DATE и TIME, в которых хранятся только составляющие даты и времени значений типа DATETIME, соответственно. Значения типа DATE занимают 3 байта, представляя диапазон дат от 01/01/0001 до 31/12/9999. Значения типа TIME занимают 3—5 байт и представляют время с точностью до 100 нс.

Тип данных DATETIME2 используется для представления значений дат и времени с высокой точностью. В зависимости от требований, значения этого типа можно определять разной длины, и занимают они от 6 до 8 байтов. Составляющая времени представляет время с точностью до 100 нс. Этот тип данных не поддерживает переход на летнее время.

Все рассмотренные на данный момент временные типы данных не поддерживают часовые пояса. Тип данных DATETIMEOFFSET имеет составляющую для хранения смещения часового пояса. По этой причине значения этого типа занимают от 6 до 8 байтов. Все другие свойства этого типа данных аналогичны соответствующим свойствам типа данных DATETIME2.

Значения дат в Transact-SQL по умолчанию определены в виде строки формата 'ммм дд гггг' (например, 'Jan 10 1993'), заключенной в одинарные или двойные кавычки. (Но относительный порядок составляющих месяца, дня и года можно изменять с помощью инструкции SET DATEFORMAT. Кроме этого, система поддерживает числовые значения для составляющей месяца и разделители / и -.) Подобным образом, значение времени указывается в 24-часовом формате в виде 'чч:мм' (например, '22:24').

Пример. Действительные представления даты

'28/5/1959' (с использованием инструкции SET DATEFORMAT dmy)

'May 28, 1959'

'1959 MAY 28'

Пример. Действительные представления времени

'8:45 AM'

'4 pm'

8.2.4. Прочие типы данных

Язык Transact-SQL поддерживает несколько типов данных, которые не принадлежат не к одной из вышеперечисленных групп типов данных. В частности:

- двоичные типы данных;
- битовый тип данных BIT;
- тип данных больших объектов;
- тип данных CURSOR;
- тип данных UNIQUEIDENTIFIER;
- тип данных SQL_VARIANT;
- тип данных TABLE;
- тип данных XML;
- пространственные типы данных;
- тип данных HIERARCHYID;
- тип данных TIMESTAMP;
- типы данных, определяемые пользователем.

8.2.4.1. Двоичные и битовые типы данных

К двоичным типам данным принадлежат два типа: BINARY и VARBINARY. Эти типы данных описывают объекты данных во внутреннем формате системы и используются для хранения битовых строк. По этой причине значения этих типов вводятся, используя шестнадцатеричные числа.

Значения битового типа BIT содержат лишь один бит, вследствие чего в одном байте можно сохранить до восьми значений этого типа. Краткое описание свойств двоичных и битовых типов данных приводится в табл. 8.3.

Таблица 8.3. Двоичные и битовые типы данных

Тип данных	Описание
BINARY[(n)]	Определяет строку битов фиксированной длины, содержащую ровно n байтов (0 < n < 8000)
VARBINARY[(n)]	Определяет строку битов переменной длины, содержащую до n байтов (0 < n < 8000)
BIT	Применяется для хранения логических значений, которые могут иметь три возможных состояния: FALSE, TRUE и NULL

8.2.4.2. Тип данных больших объектов

Тип данных LOB (Large OBject — большой объект) используется для хранения объектов данных размером до 2 Гбайт. Такие объекты обычно применяются для хранения больших объемов текстовых данных и для загрузки подключаемых модулей и аудио- и видеофайлов. В языке Transact-SQL поддерживаются следующие типы данных LOB:

- VARCHAR(max);

  NVARCHAR(max);
- VARBINARY(max).

Начиная с версии SQL Server 2005, для обращения к значениям стандартных типов данных и к значениям типов данных LOB применяется одна и та же модель программирования. Иными словами, для работы с объектами LOB можно использовать удобные системные функции и строковые операторы.

В компоненте Database Engine параметр max применяется с типами данных VARCHAR, NVARCHAR и VARBINARY для определения значений столбцов переменной длины. Когда вместо явного указания длины значения используется значение длины по умолчанию max, система анализирует длину конкретной строки и принимает решение, сохранять ли эту строку как обычное значение или как значение LOB. Параметр max указывает, что размер значений столбца может достигать максимального размера LOB данной системы.

Хотя решение о способе хранения объектов LOB принимается системой, настройки по умолчанию можно переопределить, используя системную процедуру sp_tableoption с аргументом LARGE_VALUE_TYPES_OUT_OF_ROW. Если значение этого аргумента равно 1, то данные в столбцах, объявленных с использованием параметра max, будут сохраняться отдельно от остальных данных. Если же значение аргумента равно 0, то компонент Database Engine сохраняет все значения размером до 8060 байт в строке таблицы, как обычные данные, а значения большего размера хранятся вне строки в области хранения объектов LOB.

Начиная с версии SQL Server 2008, существует два разных варианта хранения, каждый из которых позволяет сохранять объекты LOB и экономить дисковое пространство. Это следующие варианты:

- хранение данных типа FILESTREAM;
- хранение с использованием разреженных столбцов (sparse columns).

8.2.4.2.1. Хранение данных типа FILESTREAM

SQL Server поддерживает хранение больших объектов (LOB) посредством типа данных VARBINARY(max). Свойство этого типа данных таково, что большие двоичные объекты (BLOB) сохраняются в базе данных. Это обстоятельство может вызвать проблемы с производительностью в случае хранения очень больших файлов, таких как аудио- или видеофайлов. В таких случаях эти данные сохраняются вне базы данных во внешних файлах.

Хранение данных типа FILESTREAM поддерживает управление объектами LOB, которые сохраняются в файловой системе NTFS. Достоинством этого атрибута является то, что размер соответствующего объекта LOB ограничивается только размером тома файловой системы.Основным преимуществом этого типа хранения является то, что, хотя данные хранятся вне базы данных, управляются они базой данных. Таким образом, этот тип хранения имеет следующие свойства:

- данные типа FILESTREAM можно сохранять с помощью инструкции CREATE TABLE, а для работы с этими данными можно использовать инструкции для модифицирования данных (SELECT, INSERT, UPDATE и DELETE);
- система управления базой данных обеспечивает такой же самый уровень безопасности для данных типа FILESTREAM, как и для данных, хранящихся внутри базы данных.

8.2.4.2.2. Разреженные столбцы

Цель варианта хранения, предоставляемого разреженными столбцами, значительно отличается от цели хранения типа FILESTREAM. Тогда как целью хранения типа FILESTREAM является хранение объектов LOB вне базы данных, целью разреженных столбцов является минимизировать дисковое пространство, занимаемое базой данных. Столбцы этого типа позволяют оптимизировать хранение столбцов, большинство значений которых равны NULL. При использовании разреженных столбцов для хранения значений NULL дисковое пространство не требуется, но, с другой стороны, для хранения значений, отличных от NULL, требуется

дополнительно от 2 до 4 байтов, в зависимости от их типа. По этой причине разработчики Microsoft рекомендуют использовать разреженные столбцы только в тех случаях, когда ожидается, по крайней мере, 20% общей экономии дискового пространства.

Разреженные столбцы определяются таким же образом, как и прочие столбцы таблицы; аналогично осуществляется и обращение к ним. Это означает, что для обращения к разреженным столбцам можно использовать инструкции SELECT, INSERT, UPDATE и DELETE таким же образом, как и при обращении к обычным столбцам. Единственная разница касается создания разреженных столбцов: для определения конкретного столбца разреженным применяется аргумент SPARSE после названия столбца, как это показано в данном примере: имя_столбца тип_данных SPARSE

Несколько разреженных столбцов таблицы можно сгруппировать в набор столбцов. Такой набор будет альтернативным способом сохранять значения во всех разреженных столбцах таблицы и обращаться к ним.

8.2.4.3. Тип данных UNIQUEIDENTIFIER

Тип данных UNIQUEIDENTIFIER является однозначным идентификационным номером, который сохраняется в виде 16-байтовой двоичной строки. Этот тип данных тесно связан с идентификатором GUID (Globally Unique Identifier — глобально уникальный идентификатор), который гарантирует однозначность в мировом масштабе. Таким образом, этот тип данных позволяет однозначно идентифицировать данные и объекты в распределенных системах.

Инициализировать столбец или переменную типа UNIQUEIDENTIFIER можно посредством функции NEWID или NEWSEQUENTIALID, а также с помощью строковой константы особого формата, состоящей из шестнадцатеричных цифр и дефисов.

К столбцу со значениями типа данных UNIQUEIDENTIFIER можно обращаться, используя в запросе ключевое слово ROWGUIDCOL, чтобы указать, что столбец содержит значения идентификаторов. (Это ключевое слово не генерирует никаких значений.) Таблица может содержать несколько столбцов типа UNIQUEIDENTIFIER, но только один из них может иметь ключевое слово ROWGUIDCOL.

8.2.4.4. Тип данных SQL_VARIANT

Тип данных SQL_VARIANT можно использовать для хранения значений разных типов одновременно, таких как числовые значения, строки и даты. (Исключением являются значения типа TIMESTAMP.) Каждое значение столбца типа SQL_VARIANT состоит из двух частей: собственно значения и информации, описывающей это значение. Эта информация содержит все свойства действительного типа данных значения, такие как длина, масштаб и точность.

Для доступа и отображения информации о значениях столбца типа SQL_VARIANT применяется функция Transact-SQL SQL_VARIANT_PROPERTY.

8.2.4.5. Тип данных HIERARCHYID

Тип данных HIERARCHYID используется для хранения полной иерархии. Например, в значении этого типа можно сохранить иерархию всех сотрудников или иерархию папок. Этот тип реализован в виде определяемого пользователем типа CLR (Common Language Runtime — общеязыковая среда исполнения), который охватывает несколько системных функций для создания узлов иерархии и работы с ними.

Следующие функции, среди прочих, принадлежат к методам этого типа данных:
GetLevel(), GetAncestor(), GetDescendant(), Read() и Write().

8.2.4.6. Тип данных TIMESTAMP

Тип данных TIMESTAMP указывает столбец, определяемый как VARBINARY(8) или BINARY(8), в зависимости от свойства столбца принимать значения NULL. Для каждой базы данных система содержит счетчик, значение которого увеличивается всякий раз, когда вставляется или обновляется любая строка, содержащая ячейку типа TIMESTAMP, и присваивает этой ячейке данное значение. Таким образом, с помощью ячеек типа TIMESTAMP можно определить относительное время последнего изменения соответствующих строк таблицы. (ROWVERSION является синонимом TIMESTAMP.)

8.3. Функции языка SQL

Функции языка Transact-SQL могут быть агрегатными или скалярными.

8.3.1. Агрегатные функции

Агрегатные функции выполняют вычисления над группой значений столбца и всегда возвращают одно значение результата этих вычислений.

Язык Transact-SQL поддерживает несколько групп агрегатных функций. В частности, следующие:

- обычные агрегатные функции;
- статистические агрегатные функции;
- агрегатные функции, определяемые пользователем;
- аналитические агрегатные функции.

Рассмотрим следующие обычные агрегатные функции:

- функция AVG — вычисляет среднее арифметическое значение данных, содержащихся в столбце. Значения, над которыми выполняется вычисление, должны быть числовыми;
- функции MAX и MIN — определяют максимальное и минимальное значение из всех значений данных, содержащихся в столбце. Значения могут быть числовыми, строковыми или временными (дата/время);
- функция SUM — вычисляет общую сумму значений в столбце. Значения, над которыми выполняется вычисление, должны быть числовыми;
- функция COUNT — подсчитывает количество значений, отличных от NULL в столбце. Функция COUNT(*) является единственной агрегатной функцией, которая не выполняет вычисления над столбцами. Эта функция возвращает количество строк (независимо от того, содержат ли отдельные столбцы значения NULL);
- функция COUNT_BIG — аналогична функции COUNT, с той разницей, что возвращает значение данных типа BIGINT.

8.3.2. Скалярные функции

Скалярные функции Transact-SQL используются в создании скалярных выражений. (Скалярная функция выполняет вычисления над одним значением или списком значений, тогда как агрегатная функция выполняет вычисления над группой значений из нескольких строк.) Скалярные функции можно разбить на следующие категории:

- числовые функции;
- функции даты;
- строковые функции;
- системные функции;
- функции метаданных.

8.3.2.1. Числовые функции

Числовые функции языка Transact-SQL — это математические функции для модифицирования числовых значений. Список числовых функций и их краткое описание приводится в табл. 8.8.

Таблица 8.8. Числовые функции Transact-SQL

Функция	Описание
ABS(n)	Возвращает абсолютное значение (т. е. отрицательные значения возвращаются, как положительные) числового выражения n. Примеры: SELECT ABS(-5.767) = 5.767 SELECT ABS(6.384) = 6.384
ACOS(n)	Вычисляет арккосинус значения n. Исходное значение n и результат имеют тип данных FLOAT
ASIN(n)	Вычисляет арксинус значения n. Исходное значение n и результат имеют тип данных FLOAT
ATAN(n)	Вычисляет арктангенс значения n. Исходное значение n и результат имеют тип данных FLOAT
ATN2(n,m)	Вычисляет арктангенс значения n/m. Исходные значения n и m и результат имеют тип данных FLOAT
CEILING(n)	Возвращает наименьшее целое значение, большее или равное указанному параметру. Примеры: SELECT CEILING(8.88) = 5 SELECT CEILING(-8.88) = -4
COS(n)	Вычисляет косинус значения n. Исходное значение n и результат имеют тип данных FLOAT
COT(n)	Вычисляет котангенс значения n. Исходное значение n и результат имеют тип данных FLOAT
DEGREES(n)	Преобразовывает радианы в градусы. Примеры: SELECT DEGREES(PI()/2) = 90 SELECT DEGREES(0.75) = 42
EXP(n)	Вычисляет значение e^n. Пример: SELECT EXP(1) = 2.7183
FLOOR(n)	Возвращает наибольшее целое значение, меньшее или равное указанному параметру. Пример: SELECT FLOOR(8.88) = 4
LOG(n)	Вычисляет натуральный логарифм (т. е. с основанием e) числа n. Примеры: SELECT LOG(8.67) = 1.54 SELECT LOG(0.12) =-2.12
LOG10(n)	Вычисляет десятичный (с основанием 10) логарифм числа n. Примеры: SELECT LOG10(8.67) = 0.67 SELECT LOG10(0.12) =-0.92
PI()	Возвращает значение π (3,14)
POWER(x,y)	Вычисляет значение x в степени y. Примеры: SELECT POWER(3.12,5) = 295.65 SELECT POWER(81,0.5) = 9

RADIANS(n)	Преобразовывает градусы в радианы. Примеры: SELECT RADIANS(90.0) = 1.57 SELECT RADIANS(42.97) = 0.75
RAND()	Возвращает произвольное число типа FLOAT в диапазоне значений между 0 и 1
ROUND(n,p,[t])	Округляет значение n с точностью до p. Когда аргумент p положительное число, округляется дробная часть числа n, а когда отрицательное — целая часть. При использовании необязательного аргумента t≠0, число n не округляется, а усекается. Примеры: SELECT ROUND(5.4567,3) = 5.4570 SELECT ROUND(345.4567-1) = 350.0000 SELECT ROUND(345.4567-1,1) = 340.0000
ROWCOUNT_BIG	Возвращает количество строк таблицы, которые были обработаны последней инструкцией Transact-SQL, исполненной системой. Возвращаемое значение имеет тип BIGINT
SIGN(n)	Возвращает знак значения n в виде числа: +1, если положительное, –1, если отрицательное. Пример: SELECT SIGN(0.88) = 1.00
SIN(n)	Вычисляет синус значения n. Исходное значение n и результат имеют тип данных FLOAT
SQRT(n)	Вычисляет квадратный корень числа n. Пример: SELECT SQRT(9) = 3
SQUARE(n)	Возвращает квадрат аргумента n. Пример: SELECT SQUARE(9) = 81
TAN(n)	Вычисляет тангенс аргумента n. Исходное значение n и результат имеют тип данных FLOAT

8.3.2.2. Функции даты

Функции даты вычисляют соответствующие части даты или времени выражения или возвращают значение временного интервала. Поддерживаемые в Transact-SQL функции даты и их краткое описание приводятся в табл. 8.5.

Таблица 8.5. Функции даты

Функция	Описание
GETDATE()	Возвращает текущую системную дату и время. Пример: SELECT GETDATE() = 2012-03-31 13:03:31.390
DATEPART(item, date)	Возвращает указанную в параметре item часть даты date в виде целого числа. Примеры: SELECT DATEPART(month, '01.01.2005') = 3 (1 = Январь) SELECT DATEPART(weekday, '01.01.2005') = 2 (2 = Вторник)
DATENAME(item, date)	Возвращает указанную в параметре item часть даты date в виде строки символов. Пример: SELECT DATENAME(weekday, '01.01.2005') = Sanday

DATEDIFF(item, dat1, dat2)	Вычисляет разницу между двумя частями дат dat1 и dat2 и возвращает целочисленный результат в единицах, указанных в аргументе item. Пример (возвращает возраст каждого служащего): SELECT DATEDIFF(year, BirthDate, GETDATE()) AS age FROM employee
DATEADD(i, n, d)	Прибавляет n-е количество единиц, указанных в аргументе i к указанной дате d. (Значение аргумента n также может быть отрицательным.) Пример (добавляет три дня к дате приема на работу каждого служащего: SELECT DATEADD(DAY,3,HireDate) AS age FROM employee

8.3.2.3. Строковые функции

Строковые функции манипулируют значениями столбцов, которые обычно имеют символьный тип данных. Поддерживаемые в Transact-SQL строковые функции и их краткое описание приводятся в табл. 8.6.

Таблица 8.6. Строковые функции

Функция	Описание
ASCII(символ)	Преобразовывает указанный символ в соответствующее целое число кода ASCII. Пример: SELECT ASCII('A') = 65
CHAR(целое число)	Преобразовывает код ASCII в соответствующий символ. Пример: SELECT CHAR(65) = 'A'
CHARINDEX(z1,z2)	Возвращает начальную позицию вхождения подстроки z1 в строку z2. Если строка z2 не содержит подстроки z1, возвращается значение 0. Пример: SELECT CHARINDEX('bl', 'table') = 3
DIFFERENCE(z1, z2)	Возвращает целое число от 0 до 4, которое является разницей между значениями SOUNDEX двух строк z1 и z2. Метод SOUNDEX возвращает число, которое характеризует звучание строки. С помощью этого метода можно определить подобно звучащие строки. Пример: SELECT DIFFERENCE('spelling', 'telling') = 2 (звучания несколько похожи, если же функция возвращает 0, то звучания не похожи)
LEFT(z, length)	Возвращает количество первых символов строки z, заданное параметром length
LEN(z)	Возвращает количество символов (не количество байт) строки z, указанной в аргументе, включая конечные пробелы
LOWER(z1)	Преобразовывает все прописные буквы строки z1, заданной в аргументе z1, в строчные. Входящие в строку строчные буквы и иные символы не затрагиваются. Пример: SELECT LOWER('BiG') = 'big'
LTRIM(z)	Удаляет начальные пробелы в строке z. Пример: SELECT LTRIM(' String') = 'String'
NCHAR(i)	Возвращает символ в кодировке Unicode, заданный целочисленным кодом, как определено в стандарте Unicode
QUOTENAME(char_string)	Возвращает строку в кодировке Unicode с добавленными ограничителями, чтобы преобразовать строку ввода в действительный идентификатор с ограничителями

PATINDEX(%p%, expr)	Возвращает начальную позицию первого вхождения шаблона p в заданное выражение expr, или ноль, если данный шаблон не обнаружен. Примеры (второй запрос возвращает все имена из столбца custormers): SELECT PATINDEX('%gs%', 'longstring') = 4 SELECT RIGHT(ContactName, LEN(ContactName)-PATINDEX('% %', ContactName)) AS First_ name FROM Customers
REPLACE(str1,str2,str3)	Заменяет все вхождения подстроки str2 в строке str1 подстрокой str3. Пример: SELECT REPLACE('shave', 's', 'be') = behave
REPLICATE(z,i)	Повторяет i раз строку z. Пример: SELECT REPLICATE('a',10) = 'aaaaaaaaaa'
REVERSE(z)	Выводит строку z в обратном порядке. Пример: SELECT REVERSE('calculate') = 'etaluclac'
RIGHT(z,length)	Возвращает последние length-символов строки z. Пример: SELECT RIGHT('Notebook',4) = 'book'
RTRIM(z)	Удаляет конечные пробелы в строке z. Пример: SELECT RTRIM('Notebook ') = 'Notebook'
SOUNDEX(a)	Возвращает четырехсимвольный код SOUNDEX, используемый для определения похожести двух строк. Пример: SELECT SOUNDEX('spelling') = S145
SPACE(length)	Возвращает строку пробелов длиной, указанной в параметре length. Пример: SELECT SPACE(4) = ' ' (Кавычки не являются частью возвращенной строки.)
STR(f,[len[,d]])	Преобразовывает заданное выражение с плавающей точкой f в строку, где len — длина строки, включая десятичную точку, знак, цифры и пробелы (по умолчанию равно 10), а d — число разрядов дробной части, которые нужно возвратить. Пример: SELECT STR(3.45678,4,2) ='3.46'
STUFF(z1,a,length,z2)	Удаляет из строки z1 length-символов, начиная с позиции a, и вставляет на их место строку z2. Примеры: SELECT STUFF('Notebook',5,0,' in a') = 'Note in a book SELECT STUFF('Notebook',1,4, 'Hand') = 'Handbook'
SUBSTRING(z,a,length)	Извлекает из строки z, начиная с позиции a, подстроку длиной length. Пример: SELECT SUBSTRING('wardrobe',1,4) = 'ward'
UNICODE(z)	Возвращает код Unicode первого символа строки z
UPPER(z)	Преобразовывает все строчные буквы строки z в прописные. Прописные буквы и цифры не затрагиваются. Пример: SELECT UPPER('loWer') = 'LOWER'

8.3.2.4. Системные функции

Системные функции языка Transact-SQL предоставляют обширную информацию об объектах базы данных. Большинство системных функций использует внутренний числовой идентификатор (ID), который присваивается каждому объекту базы данных при его создании. Посредством этого идентификатора система может однозначно идентифицировать каждый объект базы данных. В табл. 8.7 приводятся некоторые из наиболее важных системных функций вместе с их кратким описанием.

Таблица 8.7. Системные функции

Функция	Описание
CAST(a AS type [(length)]	Преобразовывает выражение a в указанный тип данных type (если это возможно). Аргумент a может быть любым действительным выражением. Пример: SELECT CAST(3000000000 AS BIGINT) = 3000000000
COALESCE(a1,a2,...)	Возвращает первое значение выражения из списка выражений a1, a2,..., которое не является значением NULL
COL_LENGTH(obj,col)	Возвращает длину столбца col объекта базы данных (таблицы или представления) obj. Пример: SELECT COL_LENGTH('customers', 'custID') = 10
CONVERT(type[(length)],a)	Эквивалент функции CAST, но аргументы указываются по-иному. Может применяться с любым типом данных
CURRENT_TIMESTAMP	Возвращает текущие дату и время. Пример: SELECT CURRENT_TIMESTAMP = '2011-01-0117:22:55.670'
CURRENT_USER	Возвращает имя текущего пользователя
DATALENGTH(z)	Возвращает число байтов, которые занимает выражение z. Пример (возвращает длину каждого поля): SELECT DATALENGTH (ProductName) FROM products
GETANSINULL('dbname')	Возвращает 1, если использование значений NULL в базе данных dbname отвечает требованиям стандарта ANSI SQL. Пример: SELECT GETANSINULL('AdventureWorks') = 1
ISNULL(expr, value)	Возвращает значение выражения expr, если оно не равно нулю; в противном случае возвращается значение value
ISNUMERIC(expression)	Определяет, имеет ли выражение действительный числовой тип
NEWID()	Создает однозначный идентификационный номер ID, состоящий из 16-байтовой двоичной строки, предназначенной для хранения значений типа данных UNIQUEIDENTIFIER
NEWSEQUENTIALID()	Создает идентификатор GUID, больший, чем любой другой идентификатор GUID, созданный ранее этой функцией на указанном компьютере. (Эту функцию можно использовать только как значение по умолчанию для столбца.)
NULLIF(expr1,expr2)	Возвращает значение NULL, если значения выражений expr1 и expr2 одинаковые. Пример (возвращает значение NULL для проекта, для которого project_no='p1'): SELECT NULLIF(project_no, 'p1') FROM projects
SERVERPROPERTY(propertyname)	Возвращает информацию о свойствах сервера базы данных

SYSTEM_USER	Возвращает имя пользователя текущего пользователя. Пример: SELECT SYSTEM_USER = LTB13942\dusan
USER_ID([user_name])	Возвращает идентификатор пользователя user_name. Если пользователь не указан, то возвращается идентификатор текущего пользователя. Пример: SELECT USER_ID('guest') = 2
USER_NAME([id])	Возвращает имя пользователя с указанным идентификатором id. Если идентификатор не указан, то возвращается имя текущего пользователя. Пример: SELECT USER_NAME(1) = 'dbo'

Все строковые функции можно вкладывать друг в друга в любой порядке, например:

REVERSE(CURRENT_USER)

8.3.2.5. Функции метаданных

По большому счету, функции метаданных возвращают информацию об указанной базе данных и объектах базы данных. В табл. 8.8 приводятся некоторые из наиболее важных функций метаданных вместе с их кратким описанием. (Полный список всех функций метаданных см. в электронной документации.)

Таблица 8.8. Функции метаданных

Функция	Описание
COL_NAME(tab_id, col_id)	Возвращает имя столбца с указанным идентификатором col_id таблицы с идентификатором tab_id. Пример: SELECT COL_NAME(OBJECT_ID('employee'), 3) = 'emp_lname'
COLUMNPROPERTY(id, col, property)	Возвращает информацию об указанном столбце. Пример: SELECT COLUMNPROPERTY(object_id('project'), 'project_no', 'PRECISION') = 4
DATABASEPROPERTYEX (database, property)	Возвращает значение свойства property базы данных database. Пример (указывает, удовлетворяет ли база данных требованиям правил SQL-92 для разрешения значений NULL): SELECT DATABASEPROPERTYEX('sample', 'IsAnsiNullDefault') = 0
DB_ID([db_name])	Возвращает идентификатор базы данных db_name. Если имя базы данных не указано, то возвращается идентификатор текущей базы данных. Пример: SELECT DB_ID('AdventureWorks') = 6
DB_NAME([db_id])	Возвращает имя базы данных, имеющей идентификатор db_id. Если идентификатор не указан, то возвращается имя текущей базы данных. Пример: SELECT DB_NAME(6) = 'AdventureWorks'
INDEX_COL(table, i, no)	Возвращает имя индексированного столбца таблицы table. Столбец указывается идентификатором индекса i и позицией no столбца в этом индексе
INDEXPROPERTY(obj_id, index_name, property)	Возвращает свойства именованного индекса или статистики для указанного идентификационного номера таблицы, имя индекса или статистики, а также имя свойства
OBJECT_NAME(obj_id)	Возвращает имя объекта базы данных, имеющего идентификатор obj_id. Пример: SELECT OBJECT_NAME(453576654) = 'products'

OBJECT_ID(obj_name)	Возвращает идентификатор объекта obj_name базы данных. Пример: SELECT OBJECT_ID('products') = 453576654
OBJECTPROPERTY(obj_id, property)	Возвращает информацию об объектах из текущей базы данных

8.3.3. Глобальные переменные

Глобальные переменные — это специальные системные переменные, которые можно использовать, как будто бы они были скалярными константами. Язык TransactSQL поддерживает большое число глобальных переменных, именам которых предшествует префикс @@. В табл. 8.9 приводится список некоторых наиболее важных глобальных переменных и их краткое описание.

Таблица 8.9. Глобальные переменные

Переменная	Описание
@@CONNECTIONS	Возвращает число попыток входа в систему со времени запуска системы
@@CPU_BUSY	Возвращает общее время занятости центрального процессора (в миллисекундах), прошедшее с момента старта системы
@@ERROR	Возвращает информацию о возвращенном значении последней исполненной инструкции Transact-SQL
@@IDENTITY	Возвращает последнее значение, добавленное в столбец, имеющий свойство IDENTITY.
@@LANGID	Возвращает идентификатор языка, используемого в настоящий момент базой данных
@@LANGUAGE	Возвращает названия языка, используемого в настоящий момент базой данных
@@MAX_CONNECTIONS	Возвращает максимальное число фактических соединений с системой
@@PROCID	Возвращает идентификатор хранимой процедуры, исполняемой в настоящий момент
@@ROWCOUNT	Возвращает количество строк таблицы, которые были затронуты последней инструкцией Transact-SQL, исполненной системой
@@SERVERNAME	Возвращает информацию о локальном сервере базы данных. Эта информация содержит, среди прочего, имя сервера и имя экземпляра
@@SPID	Возвращает идентификатор серверного процесса
@@VERSION	Возвращает текущую версию программного обеспечения системы баз данных

8.4. Запросы

8.4.1. Инструкция Select

В языке Transact-SQL имеется одна основная инструкция для выборки информации из базы данных — инструкция SELECT. Эта инструкция позволяет извлекать информацию из одной или нескольких таблиц базы данных и даже из нескольких баз данных. Результаты выполнения инструкции SELECT помещаются в еще одну таблицу, называемую результирующим набором (result set).

Далее показан синтаксис инструкции SELECT, содержащий почти все возможные предложения:

```
SELECT select_list
  [INTO new_table]
  FROM table
  [WHERE search_condition]
  [GROUP BY group_by_expression]
```

[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]];

8.4.2. Оператор Top

Оператор TOP широко используется для ограничения числа строк, возвращаемых командой SELECT. В версиях SQL Server до 2005, оператор **TOP** принимал в качестве параметра только константу. В SQL Server 2005 и выше параметром этого оператора может быть переменная, выражение или вложенный вопрос.

Например,

следующим образом можно осуществить выборку 10 самых дешевых моделей в таблице **Products**:

SELECT TOP 10 * **FROM** Products order by [цена]

Следующим образом можно осуществить выборку такого числа моделей некоторого производителя, которое соответствует среднему количеству моделей каждого производителя в таблице **Products**:

SELECT TOP (**SELECT** AVG(AvgNum) *

FROM

(**SELECT** COUNT(*) **AS** AvgNum **FROM** Products **GROUP BY** BrandID) **AS** NumTable) P.Model

FROM Products

При использовании константы, скобки не обязательны в команде **SELECT**, но обязательны при использовании с командами, изменяющими данные, например:

DELETE TOP(10) **FROM** Orders **ORDER BY** Date **DESC**

8.4.3. Предложение Where

В предложении WHERE могут применяться следующие операторы сравнения:

=	равно
<> (или !=)	не равно
<	меньше чем
>	больше чем
>=	больше чем или равно
<=	меньше чем или равно
!>	не больше чем
!<	не меньше чем

8.4.4. Оператор LIKE

Оператор LIKE используется для сопоставления с образцом, т. е. он сравнивает значения столбца с указанным шаблоном. Столбец может быть любого символьного типа данных или типа дата. Общая форма оператора LIKE выглядит таким образом: column [NOT] LIKE 'pattern'

Параметр 'pattern' может быть строковой константой, или константой даты, или выражением (включая столбцы таблицы), и должен быть совместимым с типом данных соответствующего столбца. Для указанного столбца сравнение значения строки и шаблона возвращает TRUE, если значение совпадает с выражением шаблона.

Определенные применяемые в шаблоне символы, называющиеся подстановочными символами (wildcard characters), имеют специальное значение. Рассмотрим два из этих символов:

% (знак процента) — обозначает последовательность любых символов любой длины;

_ (символ подчеркивания) — обозначает любой один символ.

Кроме знака процентов и символа подчеркивания, поддерживает другие специальные символы, применяемые с оператором LIKE. Использование этих символов ([,], ^) демонстрируется в следующих примерах:

```
SELECT dept_nt, dept_name, location
    FROM department
    WHERE location LIKE '[C-F]%';
```

Квадратные скобки [] ограничивают диапазон или список символов. Порядок отображения символов диапазона определяется порядком сортировки, указанным при установке системы.

Символ ^ обозначает отрицание диапазона или списка символов. Но такое значение этот символ имеет только тогда, когда находится внутри квадратных скобок. В следующем примере запроса осуществляется выборка имен и фамилий тех сотрудников, чьи фамилии начинаются с буквы отличной от J, K, L, M, N, O и чьи имена начинаются не с буквы E или Z.

```
SELECT emp_no, emp_fname, emp_lname
    FROM employee
    WHERE emp_lname LIKE '[^J-O]%1'
    AND emp_fname LIKE '[^EZ]%';
```

Любой подстановочный символ (%, _, [,] или ^), заключенный в квадратные скобки, остается обычным символом и представляет сам себя. Такая же возможность существует при использовании параметра ESCAPE.

```
SELECT project_no, project_name
    FROM project
    WHERE project_name LIKE '%[_]%';


SELECT project_no, project_name
    FROM project
    WHERE project_name LIKE '%!_%' ESCAPE '!';
```

В примере обе инструкции SELECT рассматривают символ подчеркивания в значениях столбца project_name, как таковой, а не как подстановочный. В первой инструкции SELECT это достигается заключением символа подчеркивания в квадратные скобки. А во второй инструкции SELECT этот же эффект достигается за счет применения символа перехода (escape character), каковым в данном случае является символ восклицательного знака. Символ перехода переопределяет значение символа подчеркивания, делая его из подстановочного символа обычным.

8.4.5. Агрегатные функции COUNT и COUNT_BIG

Агрегатная функция COUNT имеет две разные формы:

```
COUNT([DISTINCT] col_name)
COUNT(*)
```

Первая форма функции подсчитывает количество значений в столбце col_name. Если в запросе используется ключевое слово DISTINCT, перед применением функции COUNT удаляются все повторяющиеся значения столбца. При подсчете количества значений столбца эта форма функции COUNT не принимает во внимание значения NULL.

Функция COUNT_BIG аналогична функции COUNT. Единственное различие между ними заключается в типе возвращаемого ими результата: функция COUNT_BIG всегда возвращает значения типа BIGINT, тогда как функция COUNT возвращает значения данных типа INTEGER.

8.4.6. Статистические агрегатные функции

Следующие функции составляют группу статистических агрегатных функций:

- VAR — вычисляет статистическую дисперсию всех значений, представленных в столбце или выражении;
- VARP — вычисляет статистическую дисперсию совокупности всех значений, представленных в столбце или выражении;
- STDEV — вычисляет среднеквадратическое отклонение (который рассчитывается как квадратный корень из соответствующей дисперсии) всех значений столбца или выражения;
- STDEVP — вычисляет среднеквадратическое отклонение совокупности всех значений столбца или выражения.

8.4.7. Агрегатные функции, определяемые пользователем

Компонент Database Engine также поддерживает реализацию функций, определяемых пользователем. Эта возможность позволяет пользователям дополнить системные агрегатные функции функциями, которые они могут реализовывать и устанавливать самостоятельно. Эти функции представляют специальный класс определяемых пользователем функций.

8.4.8. Использование предложения ORDER BY для разбиения результатов на страницы

Отображение результатов запроса на текущей странице можно или реализовать в пользовательском приложении, или же дать указание осуществить это серверу базы данных. В первом случае все строки базы данных отправляются приложению, чьей задачей является отобрать требуемые строки и отобразить их. Во втором случае, со стороны сервера выбираются и отображаются только строки, требуемые для текущей страницы. Как можно предположить, создание страниц на стороне сервера обычно обеспечивает лучшую производительность, т. к. клиенту отправляются только строки, необходимые для отображения.

Для поддержки создания страниц на стороне сервера в SQL Server (от версии 2012) используются два предложения инструкции SELECT: OFFSET и FETCH. Применение этих двух предложений демонстрируется в примере ниже. Здесь из базы данных извлекается идентификатор бизнеса, название должности и день рождения всех сотрудников женского пола с сортировкой результата по названию должности в возрастающем порядке. Результирующий набор строк разбивается на 10-строчные страницы и отображается третья страница.

```
SELECT BusinessEntityID, JobTitle, BirthDate
FROM HumanResources.Employee
WHERE Gender = 'F'
ORDER BY JobTitle
OFFSET 20 ROWS
FETCH NEXT 10 ROWS ONLY;
```

В предложении OFFSET указывается количество строк результата, которые нужно пропустить в отображаемом результате. Это количество вычисляется после сортировки строк предложением ORDER BY. В предложении FETCH NEXT указывается количество удовлетворяющих условию WHERE и отсортированных строк, которое нужно возвратить. Параметром этого предложения может быть константа, выражение или результат другого запроса. Предложение FETCH NEXT аналогично предложению

FETCH FIRST.

8.4.9. Инструкция SELECT и свойство IDENTITY

Свойство IDENTITY позволяет определить значения для конкретного столбца таблицы в виде автоматически возрастающего счетчика. Это свойство могут иметь столбцы численного типа данных, такого как TINYINT, SMALLINT, INT и BIGINT. Для такого столбца таблицы компонент Database Engine автоматически создает последовательные значения, начиная с указанного стартового значения. Таким образом, свойство IDENTITY можно использовать для создания однозначных числовых значений для выбранного столбца.

Таблица может содержать только один столбец со свойством IDENTITY. Владелец таблицы имеет возможность указать начальное значение и шаг приращения, как это показано в примере

```
CREATE TABLE product
    (product_no INTEGER IDENTITY(10000, 1) NOT NULL,
    product_name CHAR(30) NOT NULL,
     price MONEY);


SELECT $identity FROM product
    WHERE product_name = 'Soap';
```

Результатом этого запроса может быть следующее:

```
product_no
10005
```

В примере сначала создается таблица product, содержащая столбец product_no со свойством IDENTITY. Значения в столбце product_no создаются автоматически системой, начиная с 10 000 и увеличиваясь с единичным шагом для каждого последующего значения: 10 000, 10 001, 10 002 и т. д.

Со свойством IDENTITY связаны некоторые системные функции и переменные. Например, в коде примера используется переменная $identity. Как можно видеть по результатам выполнения этого кода, эта переменная автоматически ссылается на свойство IDENTITY.

Начальное значение и шаг приращения столбца со свойством IDENTITY можно узнать с помощью функций IDENT_SEED и IDENT_INCR соответственно. Применяются эти функции следующим образом:

```
SELECT IDENT_SEED('product'), IDENT_INCR('product)'
```

Как уже упоминалось, значения IDENTITY устанавливаются автоматически системой. Но пользователь может указать явно свои значения для определенных строк, присвоив параметру IDENTITY_INSERT значение ON перед вставкой явного значения:

```
SET IDENTITY_INSERT table_name ON
```

При вставке значений в таблицу после присвоения параметру IDENTITY_INSERT значения ON система создает следующее значение столбца IDENTITY, увеличивая наибольшее текущее значение этого столбца.

8.5. Оператор CREATE SEQUENCE

Применение свойства IDENTITY имеет несколько значительных недостатков, наиболее существенными из которых являются следующие:

- применение свойства ограничивается указанной таблицей;
- новое значение столбца нельзя получить иным способом, кроме как применив его;
- свойство IDENTITY можно указать только при создании столбца.

По этим причинам в SQL Server вводятся последовательности, которые обладают той же семантикой, что и свойство IDENTITY, но при этом не имеют ранее перечисленных недостатков. В данном контексте последовательностью называется функциональность базы данных,

позволяющая указывать значения счетчика для разных объектов базы данных, таких как столбцы и переменные.

Последовательности создаются с помощью инструкции CREATE SEQUENCE. Инструкция CREATE SEQUENCE определена в стандарте SQL и поддерживается другими реляционными системами баз данных, такими как IBM DB2 и Oracle. В примере показано создание последовательности в SQL Server.

```
CREATE SEQUENCE dbo.Sequence1
    AS INT
    START WITH 1 INCREMENT BY 5
    MINVALUE 1 MAXVALUE 256
    CYCLE;
```

Здесь значения последовательности Sequence1 создаются автоматически системой, начиная со значения 1 с шагом 5 для каждого последующего значения. Таким образом, в предложении START указывается начальное значение, а в предложении INCREMENT — шаг. (Шаг может быть, как положительным, так и отрицательным.)

В следующих двух, необязательных, предложениях MINVALUE и MAXVALUE указываются минимальное и максимальное значение объекта последовательности. В предложении CYCLE указывается, что последовательность повторяется с начала по превышению максимального (или минимального для последовательности с отрицательным шагом) значения. По умолчанию это предложение имеет значение NO CYCLE, что означает, что превышение максимального или минимального значения последовательности вызывает исключение.

Основной особенностью последовательностей является их независимость от таблиц, т. е. их можно использовать с любыми объектами базы данных, такими как столбцы таблицы или переменные. (Это свойство положительно влияет на хранение и, соответственно, на производительность. Определенную последовательность хранить не требуется; сохраняется только ее последнее значение.)

Новые значения последовательности создаются с помощью выражения NEXT VALUE FOR, применение которого показано в примере:

```
SELECT NEXT VALUE FOR dbo.Sequencel;
SELECT NEXT VALUE FOR dbo.Sequencel;
```

Результат выполнения этого запроса:

```
1
6
```

С помощью выражения NEXT VALUE FOR можно присвоить результат последовательности переменной или ячейке столбца. В примере показано использование этого выражения для присвоения результатов столбцу:

```
CREATE TABLE dbo.table1
  (column1 INT NOT NULL PRIMARY KEY,  column2 CHAR(10));
  INSERT INTO dbo.table1 VALUES (NEXT VALUE FOR dbo.sequence1, 'A');
  INSERT INTO dbo.table1 VALUES (NEXT VALUE FOR dbo.sequence1, 'B');
```

В примере сначала создается таблица table1, состоящая из двух столбцов. Далее, две инструкции INSERT вставляют в эту таблицу две строки. Первые две ячейки первого столбца будут иметь значения 11 и 16.

В следующем примере показано использование представления каталога sys.sequences для просмотра текущего значения последовательности, не используя его.

```
SELECT current_value
  FROM sys.sequences WHERE name = 'Sequence1';
```

Для изменения свойства существующей последовательности применяется инструкция ALTER SEQUENCE. Одно из наиболее важных применений этой инструкции связано с параметром RESTART WITH, который переустанавливает указанную последовательность. В следующем примере показано использование инструкции ALTER SEQUENCE для переустановки почти всех свойств последовательности Sequence1.

```
ALTER SEQUENCE dbo.sequence1
  RESTART WITH 100
  INCREMENT BY 50
  MINVALUE 50
  MAXVALUE 200
  NO CYCLE;
```

Удаляется последовательность с помощью инструкции DROP SEQUENCE.

8.6. Выражения CASE

В области прикладного программирования баз данных иногда требуется модифицировать представление данных. Например, людей можно подразделить, закодировав их по их социальной принадлежности, используя значения 1, 2 и 3, обозначив так мужчин, женщин и детей соответственно. Такой прием программирования может уменьшить время, необходимое для реализации программы. Выражение CASE языка Transact-SQL позволяет с легкостью реализовать такой тип кодировки.
Выражение CASE имеет две формы:

1. простое выражение CASE;
2. поисковое выражение CASE.

Синтаксис простого выражения CASE следующий:

```
CASE expression_1
  {WHEN expression_2 THEN result_1}...
   [ELSE result_n]
END
```

Инструкция с простым выражением CASE сначала ищет в списке всех выражений в предложении WHEN первое выражение, совпадающее с выражением expression_1, после чего выполняет соответствующее предложение THEN. В случае отсутствия в списке WHEN совпадающего выражения, выполняется предложение ELSE.

Синтаксис поискового выражения CASE следующий:

```
CASE
  {WHEN condition_1 THEN result_1} ...
    [ELSE result_n]
END
```

В данном случае выполняется поиск первого отвечающего требованиям условия, после чего выполняется соответствующее предложение THEN. Если ни одно из условий не отвечает требованиям, выполняется предложение ELSE. Применение поискового выражения CASE показано в примере:

```
SELECT project_name,
   CASE
     WHEN budget > 0 AND budget < 100000 THEN 1
     WHEN budget >= 100000 AND budget < 200000 THEN 2        WHEN budget >= 200000
AND budget < 300000 THEN 3
     ELSE 4
   END budget_weight
  FROM project;
```

Результат выполнения этого запроса:

```
project_name budget_weight
Apollo              2
Gemini              1
Mercury             2
```

В данном примере взвешиваются бюджеты всех проектов, после чего отображаются вычисленные их весовые коэффициенты вместе с соответствующими наименованиями проектов.

В следующем примере показан другой способ применения выражения CASE, где предложение WHEN содержит вложенные запросы, составляющие часть выражения.

```
SELECT project_name,
  CASE
    WHEN p1.budget < (SELECT AVG(p2.budget) FROM project p2)
      THEN 'ниже среднего'
    WHEN p1.budget = (SELECT AVG(p2.budget) FROM project p2)
      THEN 'среднее'
    WHEN p1.budget > (SELECT AVG(p2.budget) FROM project p2)
      THEN 'выше среднего'
  END budget_category
FROM project p1;
```

Результат выполнения этого запроса следующий:

```
project_name budget_category
Apollo   ниже среднего
Gemini   ниже среднего
Mercury выше среднего
```

8.7. Временные таблицы

Временная таблица — это объект базы данных, который хранится и управляется системой базы данных на временной основе. Временные таблицы могут быть локальными или глобальными. Локальные временные таблицы представлены физически, т. е. они хранятся в системной базе данных tempdb. Имена временных таблиц начинаются с префикса #, например #table_name.

Временная таблица принадлежит создавшему ее сеансу, и видима только этому сеансу. Временная таблица удаляется по завершению создавшего ее сеанса. (Также локальная временная таблица, определенная в хранимой процедуре, удаляется по завершению выполнения этой процедуры.)

Глобальные временные таблицы видимы любому пользователю и любому соединению и удаляются после отключения от сервера базы данных всех обращающихся к ним пользователей. В отличие от локальных временных таблиц имена глобальных временных таблиц начинаются с префикса ##.

В следующих примерах показано создание временной таблицы, называющейся project_temp, используя две разные инструкции языка Transact-SQL.

```
1. CREATE TABLE #project_temp   (project_no CHAR(4) NOT NULL,  project_name CHAR(25) NOT NULL);
2. SELECT project_no, project_name  INTO #project_temp1 FROM project;
```

При этом таблица, созданная в примере 1 инструкцией CREATE TABLE, остается пустой, а созданная в примере 2 инструкцией SELECT заполняется данными из таблицы project.

8.8. Табличные выражения

Табличными выражениями называются подзапросы, которые используются там, где ожидается наличие таблицы. Существует два типа табличных выражений:

- производные таблицы;
- обобщенные табличные выражения.

8.8.1. Производные таблицы

Производная таблица (derived table) — это табличное выражение, входящее в предложение FROM запроса. Производные таблицы можно применять в тех случаях, когда использование псевдонимов столбцов не представляется возможным, поскольку транслятор SQL обрабатывает другое предложение до того, как псевдоним станет известным. В примере показана попытка использовать псевдоним столбца в ситуации, когда другое предложение обрабатывается до того, как станет известным псевдоним:

```
SELECT MONTH(enter_date) as enter_month
  FROM works_on
  GROUP BY enter_month;
```

Попытка выполнить этот запрос выдаст следующее сообщение об ошибке:

```
Message 207: Level 16, State 1, Line 4 The invalid column 'enter_month'
```

Причиной ошибки является то обстоятельство, что предложение GROUP BY обрабатывается до обработки соответствующего списка инструкции SELECT, и при обработке этой группы псевдоним столбца enter_month неизвестен.

Эту проблему можно решить, используя производную таблицу, содержащую предшествующий запрос (без предложения GROUP BY), поскольку предложение FROM исполняется перед предложением GROUP BY:

```
SELECT enter_month
  FROM (SELECT MONTH(enter_date) as enter_month FROM works_on) AS m
  GROUP BY enter_month;
```

8.8.2. Обобщенные табличные выражения

Обобщенным табличным выражением (ОТВ) (Common Table Expression — сокращенно CTE) называется именованное табличное выражение, поддерживаемое языком Transact-SQL. Обобщенные табличные выражения используются в следующих двух типах запросов:

- нерекурсивных;
- рекурсивных.

8.8.2.1. ОТВ и нерекурсивные запросы

Нерекурсивную форму ОТВ можно использовать в качестве альтернативы производным таблицам и представлениям. Обычно ОТВ определяется посредством предложения WITH и дополнительного запроса, который ссылается на имя, используемое в предложении WITH.

В следующих примерах показано использование ОТВ в нерекурсивных запросах. В примере 1 используется стандартное решение, в примере 2 та же задача решается посредством нерекурсивного запроса.

Пример стандартного решения:

```
USE AdventureWorks;
  SELECT SalesOrderID
    FROM Sales.SalesOrderHeader
      WHERE TotalDue > (SELECT AVG(TotalDue)
          FROM Sales.SalesOrderHeader
          WHERE YEAR(OrderDate) = '2015')
        AND Freight > (SELECT AVG(TotalDue)
```

```
    FROM Sales.SalesOrderHeader
    WHERE YEAR(OrderDate) = '2015')/2.5;
```

Запрос в примере выбирает заказы, чьи общие суммы налогов (TotalDue) большие, чем среднее значение по всем налогам, и плата за перевозку (Freight) которых больше чем 40% среднего значения налогов. Основным свойством этого запроса является его объемистость, поскольку вложенный запрос требуется писать дважды. Одним из возможных способов уменьшить объем конструкции запроса будет создать представление, содержащее вложенный запрос. Но это решение несколько сложно, поскольку требует создания представления, а потом его удаления после окончания выполнения запроса. Лучшим подходом будет создать ОТВ:

```
USE AdventureWorks;
WITH price_calc(year_2015) AS
  (SELECT AVG(TotalDue)
    FROM Sales.SalesOrderHeader
    WHERE YEAR(OrderDate) = '2015')
SELECT SalesOrderID
  FROM Sales.SalesOrderHeader
  WHERE TotalDue > (SELECT year_2015  FROM price_calc)
AND Freight > (SELECT year_2015  FROM price_calc)/2.5;
```

Синтаксис предложения WITH в нерекурсивных запросах имеет следующий вид:

```
WITH cte_name (column_list) AS
(inner_query) outer_query
```

Параметр cte_name представляет имя ОТВ, которое определяет результирующую таблицу, а параметр column_list — список столбцов табличного выражения. (В примере ОТВ называется price_calc и имеет один столбец — year_2015.) Параметр inner_query представляет инструкцию SELECT, которая определяет результирующий набор соответствующего табличного выражения. После этого определенное табличное выражение можно использовать во внешнем запросе outer_query. (Внешний запрос использует ОТВ price_calc и ее столбец year_2015, чтобы упростить употребляющийся дважды вложенный запрос.)

8.8.2.2. ОТВ и рекурсивные запросы

Посредством ОТВ можно реализовывать рекурсии, поскольку ОТВ могут содержать ссылки на самих себя. Основной синтаксис ОТВ для рекурсивного запроса выглядит таким образом:

```
  WITH cte_name (column_list) AS
    (anchor_member UNION ALL  recur-
sive_member)    outer_query
```

Параметры cte_name и column_list имеют такое же значение, как и в ОТВ для нерекурсивных запросов. Тело предложения WITH состоит из двух запросов, объединенных оператором UNION ALL. Первый запрос вызывается только один раз, и он начинает накапливать результат рекурсии. Первый операнд оператора UNION ALL не ссылается на ОТВ. Этот запрос называется опорным запросом или источником.

Второй запрос содержит ссылку на ОТВ и представляет ее рекурсивную часть. Вследствие этого он называется рекурсивным членом. В первом вызове рекурсивной части ссылка на ОТВ представляет результат опорного запроса. Рекурсивный член использует результат первого вызова запроса. После этого система снова вызывает рекурсивную часть. Вызов рекурсивного члена прекращается, когда предыдущий его вызов возвращает пустой результирующий набор.

Оператор UNION ALL соединяет накопившиеся на данный момент строки, а также дополнительные строки, добавленные текущим вызовом рекурсивного члена. (Наличие оператора UNION ALL означает, что повторяющиеся строки не будут удалены из результата.)

Наконец, параметр outer_query определяет внешний запрос, который использует ОТВ для получения всех вызовов объединения обеих членов.

Для демонстрации рекурсивной формы ОТВ мы используем таблицу, определенную и заполненную кодом:

```
CREATE TABLE airplane
(containing_assembly VARCHAR(10),
contained_assembly VARCHAR(10),
quantity_contained INT,
unit_cost DECIMAL (6,2));
```

Таблица airplane состоит из четырех столбцов. Столбец containing_assembly определяет сборку, а столбец contained_assembly — части (одна за другой), которые составляют соответствующую сборку. На рис. 8.1 приведена графическая иллюстрация возможного вида самолета и его составляющих частей.

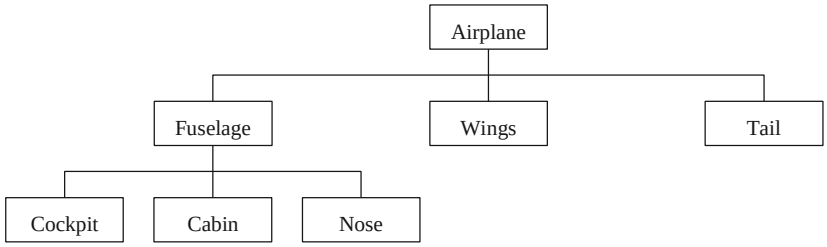

Рис. 8.1. Графическое представление самолета и его составляющих

Таблица airplane содержит 11 строк, которые показаны в табл. 8.10.

Таблица 8.10. Содержимое таблицы airplane

Airplane	Fuselage	1	10
Airplane	Wings	1	11
Airplane	Tail	1	12
Fuselage	Cockpit	1	13
Fuselage	Cabin	1	14
Fuselage	Nose	1	15
Cockpit	NULL	1	13
Cabin	NULL	1	14
Nose	NULL	1	15
Wings	NULL	2	11
Tail	NULL	1	12

В примере показано применение предложения WITH для определения запроса, который вычисляет общую стоимость каждой сборки.

```
WITH list_of_parts(assembly1, quantity, cost) AS
  (SELECT containing_assembly, quantity_contained, unit_cost
```

```
    FROM airplane
    WHERE contained_assembly IS NULL
  UNION ALL
  SELECT a.containing_assembly, a.quantity_contained,
  CAST(1.quantity*1.cost AS DECIMAL(6,2))
  FROM list_of_parts 1, airplane a
  WHERE 1.assembly1 = a.contained_assembly)
SELECT * FROM list_of_parts;
```

Предложение WITH определяет список ОТВ с именем list_of_parts, состоящий из трех столбцов: assembly, quantity и cost. Первая инструкция SELECT в примере вызывается только один раз, чтобы сохранить результаты первого шага процесса рекурсии.

Инструкция SELECT в последней строке примера отображает следующий результат:

Assembly	quantity	costs
Cockpit	1	13.00
Cabin	1	18.00
Nose	1	16.500
Wings	2	11.00
Tail	1	12.00
Airplane	1	12.00
Airplane	1	22.00
Fuselage	1	16.500
Airplane	1	16.500
Fuselage	1	18.00
Airplane	1	18.00
Fuselage	1	13.00
Airplane	1	13.00

Первые пять строчек этого результата являются результирующим набором первого вызова опорного члена запроса, а все остальные строчки — результатом рекурсивного члена (вторая часть) запроса в этом примере. Рекурсивный член запроса вызывается дважды: первый раз для сборки фюзеляжа (fuselage), а второй раз для всего самолета (airplane).

Запрос в следующем примере вычисляет стоимость каждой сборки со всеми ее составляющими.

```
WITH list_of_parts(assembly, quantity, cost) AS
  (SELECT containing_assembly, quantity_contained, unit_cost
  FROM airplane
  WHERE contained_assembly IS NULL
UNION ALL
SELECT a.containing_assembly, a.quantity_contained,
  CAST(1.quantity*1.cost AS DECIMAL(6,2))
  FROM list_of_parts 1, airplane a
  WHERE 1.assembly = a.contained_assembly)
SELECT assembly, SUM(quantity) parts, SUM(cost) sum_cost
  FROM  list_of_parts
  GROUP BY assembly;
```

Результат выполнения этого запроса дает следующий результат:

Assembly	parts	sum_cost
Airplane	5	76.00

Cabin	1	18.00
Cockpit	1	13.00
Fuselage	3	42.00
Nose	1	16.500
Tail	1	12.00
Wings	2	11.00

В рекурсивных запросах на ОТВ налагается несколько следующих ограничений:

- определение ОТВ должно содержать, по крайней мере, две инструкции SELECT (опорный член и рекурсивный член), объединенные оператором UNION ALL;
- опорный член и рекурсивный член должны иметь одинаковое количество столбцов (это является прямым следствием использования оператора UNION ALL);
- столбцы в рекурсивном члене должны иметь такой же тип данных, как и соответствующие столбцы в опорном члене;
- предложение FROM рекурсивного члена должно ссылаться на имя ОТВ только один раз;
- определение рекурсивного члена не может содержать следующие параметры: SELECT DISTINCT, GROUP BY, HAVING, агрегатные функции, TOP и подзапросы. Кроме этого, единственным типом операции соединения, разрешенной в определении запроса, является внутреннее соединение.