Титульный лист материалов по дисциплине

(заполняется по каждому виду учебного материала)

ДИСЦИПЛИНА **Безопасность систем баз данных**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **Кибербезопасности и цифровых технологий**

КАФЕДРА **«Информационно-аналитические системы кибербезопасности»**

(полное наименование кафедры)

ВИД УЧЕБНОГО МАТЕРИАЛА **Практическое занятие №2**

ПРЕПОДАВАТЕЛЬ **Шукенбаев А.Б., Войтенков Д.В.**

(фамилия, имя, отчество)

СЕМЕСТР **Весенний семестр, 2024-2025**

(указать семестр обучения, учебный год)

Москва 2025

Практическая работа 2. Язык Transact-SQL. Компоненты SQL

- **Основные объекты SQL**
- **Типы данных**
- **Функции языка SQL**
- **Скалярные операторы**
- **Значения *NULL***

Рассмотрим основные объекты и базовые операторы языка Transact-SQL. Сначала рассматрим базовые элементы языка, включая константы, идентификаторы и ограничители. Далее, поскольку каждый простой объект имеет соответствующий тип данных, подробно рассматрим типы данных. Кроме этого, объясняются все операторы и функции. В конце представлены значения NULL.

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.

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. В примерах 1 и 2 представлена иллюстрация некоторых допустимых и недопустимых строковых и шестнадцатеричных констант.

Пример 1. Допустимые строковые и шестнадцатеричные константы

‘Philadelphia’
“Berkeley, CA 94710”
‘9876’
0x53514C0D
’Апостроф представляется таким образом: can’’t’

(Обратите внимание в последней строке примера на использование двух последовательных одинарных кавычек для представления апострофа в строковых константах.)

Пример 2. Недопустимые строковые константы

’AB’C’ — нечетное число одинарных кавычек;
’New York" — строка должна быть заключена в одинаковые кавычки, т. е. одинарные или двойные с каждого конца строки.

К числовым константам относятся все целочисленные значения и значения с фиксированной и плавающей запятой (точкой). В примере 3 иллюстрируется несколько числовых констант.

Пример 3. Числовые константы

-130.00

-0.357E5 — экспоненциальное представление, где nEm означает n умножено на 10 в степени m
22.3E-3

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). (Типы данных разных видов литералов рассматриваются в *этой работе далее.)*

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.

ПРИМЕЧАНИЕ

Различие между одинарными и двойными кавычками было впервые введено в стандарте SQL92. Применительно к идентификаторам, этот стандарт различает между обычными идентификаторами и идентификаторами с ограничителями. Два ключевых отличия состоят в том, что идентификаторы с ограничителями заключаются в двойные кавычки и чувствительны к регистру. (В языке Transact-SQL также поддерживается применение квадратных скобок вместо двойных кавычек.) Двойные кавычки применяются только в качестве ограничителя строк. По большому счету, идентификаторы с ограничителями были введены для того, чтобы позволить определять идентификаторы, которые иначе идентичны зарезервированным ключевым словам. В частности, идентификаторы с ограничителями предотвращают использование имен (идентификаторов и имен переменных), которые могут быть введены в качестве зарезервированных ключевых слов в будущих стандартах SQL. Кроме этого, идентификаторы с ограничителями могут содержать символы, которые, как правило, не разрешаются в именах обычных идентификаторов, например, пробелы.

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.

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.)

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.

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).

ПРИМЕЧАНИЕ

В языке Transact-SQL имена всех типов данных и системных функций, такие как **character** и **integer,** не являются зарезервированными ключевыми словами. Поэтому такие слова можно использовать для обозначения объектов. Тем не менее настоятельно рекомендуется не делать этого, т. к. такая практика влечет за собой трудности в чтении и понимании инструкций Transact-SQL.

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.

Числовые типы данных

Как и следовало ожидать по их названию, числовые типы данных применяются для представления чисел. Эти типы и их краткое описание приводятся в табл. 1.

Таблица 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.
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

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. В табл. 2 перечислены категории символьных типов данных с их кратким описанием.

Таблица 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

ПРИМЕЧАНИЕ

Тип данных **varchar** идентичен типу данных **char,** за исключением одного различия: если содержимое строки **CHAR(n)** короче, чем **n** символов, остаток строки заполняется пробелами. А количество байтов, занимаемых строкой типа **varchar,** всегда равно количеству символов в ней.

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').

ПРИМЕЧАНИЕ

Язык Transact-SQL поддерживает различные форматы ввода значений типа **datetime.** Как уже упоминалось, каждая составляющая определяется отдельно, поэтому значения дат и времени можно указать в любом порядке или отдельно. Если одна из составляющих не указывается, система использует для него значение по умолчанию. (Значение по умолчанию для времени — 12:00 AM (до полудня).)

В примерах 4 и 5 показаны разные способы представления значений даты и времени в разных форматах.

Пример 4. Действительные представления даты

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

Пример 5. Действительные представления времени

'8:45 AM'
'4 pm'

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.

Рассмотрим некоторые из них

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. 3.

Таблица 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,

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 8 060 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, для столбцов типа VARBiNARY(max) можно применять атрибут filestream, чтобы сохранять данные BLOB (Binary Large OBject — большой двоичный объект) непосредственно в файловой системе NTFS. Основным достоинством этого атрибута является то, что размер соответствующего объекта LOB ограничивается только размером тома файловой системы.

Тип данных *UNIQUEIDENTIFIER*

Как можно судить по его названию, 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.

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.

> *ПРИМЕЧАНИЕ*
>
> Объявлять тип столбца как **sql_variant** следует только в том случае, если это действительно необходимо. Например, если столбец предназначается для хранения значений разных типов данных или если при создании таблицы тип данных, которые будут храниться в данном столбце, неизвестен.

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() .

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.)

> *ПРИМЕЧАНИЕ*
>
> Само по себе значение, сохраняемое в столбце типа **timestamp,** не представляет никакой

важности. Этот столбец обычно используется для определения, изменилась ли определенная строка таблицы со времени последнего обращения к ней.

Варианты хранения

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).

Хранение данных типа *FILESTREAM*

Как уже упоминалось ранее, 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. 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.

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. Дополнительную информацию
о наборах столбцов см. в ***электронной документации.***

Функции языка SQL

Функции языка Transact-SQL могут быть агрегатными или скалярными.

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.

Далее мы будем рассматривать только 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.

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.

Числовые функции

Числовые функции языка Transact-SQL — это математические функции для модифицирования числовых значений. Список числовых функций и их краткое описание приводится в табл. 4.

Таблица 4. 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(4.88) = 5 SELECT CEILING(-4.88) = -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:
LOG(n)	Вычисляет натуральный логарифм (т. е. с основанием е) числа n. Примеры: SELECT LOG(4.67) = 1.54 SELECT LOG (0.12) =-2.12
LOG1O(n)	Вычисляет десятичный (с основанием 10) логарифм числа n. Примеры: SELECT LOG10(4.67) = 0.67 SELECT LOG10(0.12) =-0.92
PI()	Возвращает значение п (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

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. 5.

Таблица 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, datl, dat2)***	Вычисляет разницу между двумя частями дат ***datl*** 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; см. базу данных sample): SELECT DATEADD(DAY,3,HireDate) AS age FROM employee

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. 6.

Таблица 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 zl 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(zl)	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 (***strl, str2, str3)***	Заменяет все вхождения подстроки ***str2*** в строке ***strl*** 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 S PACE(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 (zl, a, ***length, z2)***	Удаляет из строки ***zl 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'

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. 7 приводятся некоторые из наиболее важных системных функций вместе с их кратким описанием.

Таблица 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(al,a2,...)	Возвращает первое значение выражения из списка выражений al, 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. При-

	мер (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 (*exprl, expr2)*	Возвращает значение NULL, если значения выражений *exprl* 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)

Функции метаданных

По большому счету, функции метаданных возвращают информацию об указанной базе данных и объектах базы данных. В табл. 8 приводятся некоторые из наиболее важных функций метаданных вместе с их кратким описанием.

Таблица 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

Скалярные операторы

Скалярные операторы используются для работы со скалярными значениями. Язык Transact-SQL поддерживает числовые и логические операторы, а также конкатенацию.

Существуют унарные и бинарные арифметические операторы. Унарными операторами являются знаки + и -. К бинарным арифметическим операторам относятся операторы сложения (+), вычитания (-), умножения (*), деления (/) и деления по модулю (%).

Логические операторы обозначаются двумя разными способами, в зависимости от того, применяются они к битовым строкам или к другим типам данных. Операторы not, and и or применяются со всеми типами данных (за исключением данных типа bit).

Далее описаны побитовые (логические) операторы для обработки строк, а в примере 6 демонстрируется их использование.

- ~ — логическое отрицание — инверсия (т. е. оператор not);
- & — конъюнкция битовых строк (т. е. оператор AND);
- | — дизъюнкция битовых строк (т. е. оператор OR);
- ^ — исключающая дизъюнкция (т. е. оператор XOR или "Исключающее OR").

Пример 6. Применение побитовых операторов для манипулирования битовыми строками

-(1001001) = (0110110)
(11001001) | (10101101) = (11101101)
(11001001)&(10101101) = (10001001)
(11001001) A (10101101) = (01100100)

Оператор конкатенации + применяется для соединения символьных или битовых строк.

Глобальные переменные

Глобальные переменные — это специальные системные переменные, которые можно использовать, как будто бы они были скалярными константами. Язык Transact- SQL поддерживает большое число глобальных переменных, именам которых предшествует префикс @@. В табл. 9 приводится список некоторых наиболее важных глобальных переменных и их краткое описание.

Таблица 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

Значение NULL

Значение NULL — это специальное значение, которое можно присвоить ячейке таблицы. Это значение обычно применяется, когда информация в ячейке неизвестна или неприменима. Например, если неизвестен номер домашнего телефона служащего компании, рекомендуется присвоить соответствующей ячейке столбца
home_telephone значение NULL.

Если значение любого операнда любого арифметического выражения равно NULL, значение результата вычисления этого выражения также будет NULL. Поэтому в

унарных арифметических операциях, если значение выражения A равно NULL, тогда как +A, так и -A возвращает NULL. В бинарных выражениях, если значение одного или обоих операндов A и B равно NULL, тогда результат операции сложения (A+B), вычитания (A-B), умножения (A*B), деления (A/B) и деления по модулю (A%B) также будет NULL. (Операнды A и B должны быть числовыми выражениями.)

Если выражение содержит операцию сравнения и значение одного или обоих операндов этой операции равно NULL, результат этой операции также будет NULL. Следовательно, в таком случае все выражения: A=B, A<>B, A<B и A>B — возвратят значение NULL.

Для логических операций and, or и not поведение значений null описывается в следующих ***таблицах истинности,*** где и означает истина (true), н — неизвестно (null), а л — ложь (false). В этих таблицах, операнды логических операторов and и or представлены как заглавия столбцов и строк, а результат операции для конкретной пары операндов находится в ячейке на пресечении соответствующего столбца и строки. Для оператора not операнды представлены в столбце "Операнд", а результат в ячейке справа.

Операция and			
	И	**Н**	**Л**
И	И	Н	Л
Н	Н	Н	Л
Л	Л	Л	Л

Операция OR			
	И	**Н**	**Л**
И	И	И	И
Н	И	Н	Н
Л	И	Н	Л

Операция not	
Операнд	
И	Л
Н	Н
Л	И

Перед вычислением агрегатных функций AVG, SUM, MAX, MIN и COUNT (за исключением функции COUNT(*)) в их аргументах удаляются все значения NULL. Если все ячейки столбца содержат только значения NULL, функция возвращает NULL. Агрегатная функция COUNT(*) обрабатывает все значения NULL таким же образом, как и значения, не являющиеся NULL. Если столбец содержит только значения NULL, функция COUNT(DISTINCT *column_name)* возвращает 0.

Значение NULL должно отличаться от всех других значений. Для числовых типов данных значение о и значение NULL не являются одинаковыми. То же самое относится и к пустой строке и значению NULL для символьных типов данных.

Значения NULL можно сохранять в столбце таблицы только в том случае, если это явно разрешено в определении данного столбца. С другой стороны, значения NULL не разрешаются для столбца, если в его определении явно указано NOT NULL. Если для столбца с типом данных (за исключением типа TIMESTAMP) не указано явно NULL или NOT NULL, то присваиваются следующие значения:

- NULL, если значение параметра ANSI_NULL_DFLT_ON инструкции SET равно ON.
- NOT NULL, если значение параметра ANSI_NULL_DFLT_OFF инструкции SET равно ON.

Если инструкцию SET не активировать, то столбец по умолчанию будет содержать значение NOT NULL. (Для столбцов типа TIMESTAMP значения NULL не разрешаются.)

Упражнения

Упражнение 1

Какая разница между числовыми типами данных INT, SMALLINT и TINYINT?

Упражнение 2

Какая разница между типами данных CHAR и VARCHAR? Когда следует использовать первый, а не второй, и наоборот?

Упражнение 3

Как настроить столбец типа данных DATE для ввода значений в формате 'гггг/мм/дд'?

В следующих двух упражнениях используйте инструкцию SELECT в окне редактора запросов средства Management Studio для вывода результатов всех системных функций и глобальных переменных. (Например, инструкция SELECT host_id () выводит идентификационный номер текущего хоста.)

Упражнение 4

Используя системные функции, узнайте идентификационный номер базы данных test, которую создали на прошлой лабораторной работе.

Упражнение 5

Используя системные переменные, узнайте текущую версию программного обеспечения системы базы данных и используемый в программном обеспечении язык.

Упражнение 6

Используя битовые операторы &, | и ^, выполните следующие операции над битовыми строками:
(11100101) & (01010111)
(10011011) | (11001001)
(10110111) ^ (10110001)

Упражнение 7

Какими будут результаты следующих выражений? (Выражение A — числовое, а B — логическое.)
A + NULL

NULL = NULL

B OR NULL

B AND NULL

Упражнение 8

В каких случаях можно использовать как одинарные, так и двойные кавычки для определения строковых и временных констант?

Упражнение 9

Что такое идентификатор с ограничителями и когда требуется использовать идентификаторы этого типа?