Типы данных языка SQL, определенные стандартом

Данные – это совокупная информация, хранимая в базе *данных* в виде одного из нескольких различных *типов* . С помощью ***типов данных*** устанавливаются основные правила для *данных*, содержащихся в конкретном столбце таблицы, в том числе размер выделяемой для них памяти.

В языке *SQL* имеется шесть скалярных *типов данных*, определенных стандартом. Их краткое описание представлено в таблице.

Таблица 2.1.

Тип данных	**Объявления**
Символьный	`CHAR` | `VARCHAR`
Битовый	`BIT` | `BIT VARYING`
Точные числа	`NUMERIC` | `DECIMAL` | `INTEGER` | `SMALLINT`
Округленные числа	`FLOAT` | `REAL` | `DOUBLE PRECISION`
Дата/время	`DATE` | `TIME` | `TIMESTAMP`
Интервал	`INTERVAL`

Символьные данные

Символьные данные состоят из последовательности символов, входящих в определенный создателями СУБД набор символов. Поскольку наборы символов являются специфическими для различных диалектов языка SQL, перечень символов, которые могут входить в состав значений *данных символьного типа*, также зависит от конкретной реализации. Чаще всего используются наборы символов ASCII и *EBCDIC*. Для определения *данных символьного типа* используется следующий формат:

```
<символьный_тип>::=
{ CHARACTER [ VARYING][длина] | [CHAR |
VARCHAR][длина]}
```

При определении столбца с *символьным типом данных* параметр длина применяется для указания максимального количества символов, которые могут быть помещены в данный столбец (по умолчанию принимается значение `1`). Символьная строка может быть определена как имеющая фиксированную или переменную (`VARYING`) длину. Если строка определена с фиксированной длиной значений, то при вводе в нее меньшего количества символов значение дополняется до указанной длины пробелами, добавляемыми справа. Если строка определена с переменной длиной значений, то при вводе в нее меньшего количества символов в базе *данных* будут сохранены только введенные символы, что позволит достичь определенной экономии внешней памяти.

Битовые данные

Битовый тип *данных* используется для определения битовых строк, т.е. последовательности двоичных цифр (битов), каждая из которых может иметь значение либо `0` , либо `1` . *Данные битового типа* определяются при помощи следующего формата:

```
<битовый_тип>::=
BIT [VARYING][длина]
```

Точные числа

Тип точных числовых данных применяется для определения чисел, которые имеют точное представление, т.е. числа состоят из цифр, необязательной десятичной точки и необязательного символа знака. *Данные точного числового типа* определяются точностью и длиной дробной части. Точность задает общее количество значащих десятичных цифр числа, в которое входит длина как целой части, так и дробной, но без учета самой десятичной точки. Масштаб указывает количество дробных десятичных разрядов числа.

```
<фиксированный_тип>::=
{NUMERIC[точность[,масштаб]]|{DECIMAL|DEC}
        [точность[, масштаб]]
| {INTEGER |INT}| SMALLINT}
```

Типы `NUMERIC` и `DECIMAL` предназначены для хранения чисел в десятичном формате. По умолчанию длина дробной части равна нулю, а принимаемая по умолчанию точность зависит от реализации. *Тип* `INTEGER` (`INT`) используется для хранения больших положительных или отрицательных целых чисел. *Тип* `SMALLINT` – для хранения небольших положительных или отрицательных целых чисел; в этом случае расход внешней памяти существенно сокращается.

Округленные числа

Тип округленных чисел применяется для описания *данных*, которые нельзя точно представить в компьютере, в частности действительных чисел. *Округленные числа* или числа с плавающей точкой представляются в научной нотации, при которой число записывается с помощью мантиссы, умноженной на определенную степень десяти (порядок), например: `10E3` , `+5.2E6` , `-0.2E-4` . Для определения *данных* вещественного *типа* используется формат:

```
<вещественный_тип>::=
{ FLOAT [точность]| REAL |
        DOUBLE PRECISION}
```

Параметр `точность` задает количество значащих цифр мантиссы. Точность *типов* `REAL` и `DOUBLE PRECISION` зависит от конкретной реализации.

Дата и время

Тип данных "дата/время" используется для определения моментов времени с некоторой установленной точностью. Стандарт SQL поддерживает следующий формат:

```
<тип_даты/времени>::=
{DATE | TIME[точность][WITH TIME ZONE]|
   TIMESTAMP[точность][WITH TIME ZONE]}
```

Тип данных `DATE` используется для хранения календарных дат, включающих поля `YEAR` (год), `MONTH` (месяц) и `DAY` (день). *Тип данных* `TIME` – для хранения отметок времени, включающих поля `HOUR` (часы), `MINUTE` (минуты) и `SECOND` (секунды). *Тип данных* `TIMESTAMP` – для совместного хранения даты и времени. Параметр `точность` задает количество дробных десятичных знаков, определяющих точность сохранения значения в поле `SECOND` . Если этот параметр опускается, по умолчанию его значение для столбцов *типа* `TIME` принимается равным нулю (т.е. сохраняются целые секунды), тогда как для полей *типа* `TIMESTAMP` он принимается равным `6` . Наличие ключевого слова `WITH TIME ZONE` определяет использование полей `TIMEZONE HOUR` и `TIMEZONE MINUTE` , тем самым задаются час и минуты сдвига зонального времени по отношению к универсальному координатному времени (Гринвичскому времени).

Данные типа `INTERVAL` используются для представления периодов времени.

Понятие домена

Домен – это набор допустимых значений для одного или нескольких атрибутов. Если в таблице базы *данных* или в нескольких таблицах присутствуют столбцы, обладающие одними и теми же характеристиками, можно описать *тип* такого столбца и его поведение через *домен*, а затем поставить в соответствие каждому из одинаковых столбцов имя *домена*. *Домен* определяет все потенциальные значения, которые могут быть присвоены атрибуту.

Стандарт SQL позволяет определить *домен* с помощью следующего *оператора*:

```
<определение_домена>::=
        CREATE DOMAIN имя_домена [AS]
                тип_данных
        [ DEFAULT значение]
        [ CHECK (допустимые_значения)]
```

Каждому создаваемому *домену* присваивается имя, *тип данных*, значение по умолчанию и набор допустимых значений. Следует отметить, что приведенный формат *оператора* является неполным. Теперь при создании таблицы можно указать вместо *типа данных* имя *домена*.

Удаление *доменов* из базы *данных* выполняется с помощью *оператора*:

```
DROP DOMAIN имя_домена [ RESTRICT |
        CASCADE]
```

В случае указания ключевого слова `CASCADE` любые столбцы таблиц, созданные с использованием удаляемого *домена*, будут автоматически изменены и описаны как содержащие *данные* того *типа*, который был указан в определении удаляемого *домена*.

Альтернативой *доменам* в среде SQL Server являются *пользовательские типы данных* .

Типы данных, используемые в SQL-сервере

Системные типы данных

Один из основных моментов процесса создания таблицы – определение *типов данных* для ее полей. *Тип данных* поля таблицы определяет тип информации, которая будет размещаться в этом поле. Понятие *типа данных* в SQL Server полностью адекватно понятию *типа данных* в современных языках программирования. SQL-сервер поддерживает большое число различных *типов данных*: текстовые, числовые, двоичные (см. таблицу 2.2).

Таблица 2.2.

`image`	`smalldatetime`	`bit`	`binary`
`text`	`real`	`decimal`	`char`
`uniqueidentifier`	`money`	`numeric`	`timestamp`
`tinyint`	`datetime`	`smallmoney`	`nvarchar`
`smallint`	`float`	`varbinary`	`nchar`
`int`	`ntext`	`varchar`	`sysname`

Приведем краткий обзор *типов данных* SQL Server.

Для хранения символьной информации используются *символьные типы данных*, к которым относятся `CHAR` (длина), `VARCHAR` (длина), `NCHAR` (длина), `NVARCHAR` (длина). Последние два предназначены для хранения символов Unicode. Максимальное значение длины ограничено `8000` знаками (`4000` – для символов Unicode).

Хранение символьных *данных* большого объема (до 2 Гб) осуществляется при помощи текстовых *типов данных* `TEXT` и `NTEXT` .

К *целочисленным типам данных* относятся `INT` (`INTEGER`), `SMALLINT` , `TINYINT` , `BIGINT` . Для хранения *данных целочисленного типа* используется, соответственно, 4 байта (диапазон от -2^{31} до $2^{31}-1$), 2 байта (диапазон от -2^{15} до $2^{15}-1$), 1 байт (диапазон от `0` до `255`) или 8 байт (диапазон от -2^{63} до $2^{63}-1$). Объекты и *выражения целочисленного типа* могут применяться в любых математических операциях.

Числа, в составе которых есть десятичная точка, называются нецелочисленными. *Нецелочисленные данные* разделяются на два *типа* – десятичные и приблизительные.

К десятичным *типам данных* относятся *типы* `DECIMAL [(точность[,масштаб])]` или `DEC` и `NUMERIC [(точность[,масштаб])]` . *Типы данных* `DECIMAL` и `NUMERIC` позволяют самостоятельно определить формат точности числа с плавающей запятой. Параметр `точность` указывает максимальное количество цифр вводимых *данных* этого *типа* (до и после десятичной точки в сумме), а параметр `масштаб` – максимальное количество цифр, расположенных после десятичной точки. В обычном режиме сервер позволяет вводить не более 28 цифр, используемых в *типах* `DECIMAL` и `NUMERIC` (от 2 до 17 байт).

К приблизительным *типам данных* относятся `FLOAT` (точность до 15 цифр, 8 байт) и `REAL` (точность до 7 цифр, 4 байта). Эти *типы* представляют *данные* в формате с плавающей запятой, т.е. для представления чисел используется мантисса и порядок, что обеспечивает одинаковую точность вычислений независимо от того, насколько мало или велико значение.

Для хранения информации о дате и времени предназначены такие *типы данных*, как `DATETIME` и `SMALLDATETIME` , использующие для представления даты и времени 8 и 4 байта соответственно.

Типы данных `MONEY` и `SMALLMONEY` делают возможным хранение информации *денежного типа* ; они обеспечивают точность значений до 4 знаков после запятой и используют 8 и 4 байта соответственно.

Тип данных `BIT` позволяет хранить один бит, который принимает значения `0` или `1` .

В среде SQL Server реализован ряд *специальных типов данных* .

Тип данных `TIMESTAMP` применяется в качестве индикатора изменения версии строки в пределах базы *данных*.

Тип данных `UNIQUEIDENTIFIER` используется для хранения глобальных уникальных идентификационных номеров.

Тип данных `SYSNAME` предназначен для идентификаторов *объектов*.

Тип данных `SQL_VARIANT` позволяет хранить значения любого из поддерживаемых SQL Server *типов данных* за исключением `TEXT` , `NTEXT` , `IMAGE` и `TIMESTAMP` .

Тип данных `TABLE` , подобно временным таблицам, обеспечивает хранение набора строк, предназначенных для последующей обработки. *Тип данных* `TABLE` может применяться только для определения локальных *переменных* и возвращаемых пользовательскими функциями значений. Пример использования *типа данных* `TABLE` приведен в лекции, посвященной функциям пользователя.

Тип данных `CURSOR` нужен для работы с такими *объектами*, как курсоры, и может быть востребован только для *переменных* и *параметров хранимых процедур* . Курсоры SQL Server представляют собой механизм обмена *данными* между сервером и клиентом. Курсор позволяет клиентским приложениям работать не с полным набором *данных*, а лишь с одной или несколькими строками. Примеры использования *данных типа* `CURSOR` мы рассмотрим в лекциях, посвященных курсорам и хранимым процедурам.

Создание пользовательского типа данных

В системе SQL-сервера имеется поддержка ***пользовательских типов данных*** . Они могут использоваться при определении какого-либо специфического или часто употребляемого формата.

Создание *пользовательского типа данных* осуществляется выполнением системной процедуры:

```
sp_addtype [@typename=]type,[@phystype=]
        system_data_type
           [,[@nulltype=]’null_type’]
```

Тип данных `system_data_type` выбирается из следующей таблицы.

Таблица 2.3.

image	smalldatetime	decimal	bit
text	real	‘decimal[(p[,s])]’	‘binary(n)’
uniqueidentifier	datetime	numeric	‘char(n)’
smallint	float	‘numeric[(p[,s])]’	‘nvarchar(n)’
int	‘float(n)’	‘varbinary(n)’	
	ntext	‘varchar(n)’	‘nchar(n)’

```
EXEC sp_addtype bir, DATETIME, 'NULL'
или
EXEC sp_addtype bir, DATETIME, ‘NOT NULL’

2.1. Создание пользовательского типа данных bir.

CREATE TABLE tab
(id_n   INT IDENTITY(1,1) PRIMARY KEY,
names   VARCHAR(40),
birthday        BIR)

2.2. Использование пользовательского типа данных bir при создании таблицы.
```

Удаление *пользовательского типа данных* происходит в результате выполнения процедуры `sp_droptype type: EXEC sp_droptype 'bir'`

Получение информации о типах данных

Получить список всех *типов данных*, включая *пользовательские*, можно из системной таблицы `systypes` :

```
SELECT * FROM  systypes
```

Преобразование типов

Нередко требуется ***конвертировать*** значения одного *типа* в значения другого. Наиболее часто выполняется *конвертирование* чисел в символьные *данные* и наоборот, для этого используется специализированная функция `STR` . Для выполнения других *преобразований* SQL Server предлагает универсальные функции `CONVERT` и `CAST` , с помощью которых значения одного *типа* преобразовываются в значения другого *типа*, если такие изменения вообще возможны. `CONVERT` и `CAST` примерно одинаковы и могут быть взаимозаменяемыми.

```
CAST(выражение AS тип_данных)
CONVERT(тип_данных[(длина)],
        выражение [,стиль])
```

С помощью аргумента `стиль` можно управлять стилем представления значений следующих *типов данных*: *дата/время*, *денежный* или *нецелочисленный*.

```
DECLARE @d DATETIME
DECLARE  @s CHAR(8)
SET @s='29.10.01'
SET @d=CAST(@s AS DATETIME)

2.3. Преобразование данных символьного типа к данным типа дата/время.
```

Наряду с *типами данных* основополагающими понятиями при работе с языком SQL в среде MS SQL Server являются *выражения*, *операторы*, *переменные*, *управляющие конструкции* .

Выражения

Выражения представляют собой комбинацию идентификаторов, функций, знаков логических и арифметических операций, констант и других *объектов*. *Выражение* может быть использовано в качестве аргумента в командах, хранимых процедурах или запросах.

Выражение состоит из *операндов* (собственно *данных*) и *операторов* (знаков операций, производимых над *операндами*). В качестве *операндов* могут выступать *константы*, *переменные*, имена столбцов, функции, подзапросы.

Операторы – это знаки операций над одним или несколькими *выражениями* для создания нового *выражения*. Среди *операторов* можно выделить унарные *операторы*, *операторы* присваивания, арифметические *операторы*, строковые *операторы*, *операторы* сравнения, логические *операторы*, битовые *операторы* .

Переменные

В среде *SQL Server* существует несколько способов передачи *данных* между командами. Один из них – передача *данных* через локальные *переменные*. Прежде чем использовать какую-либо *переменную*, ее следует объявить. Объявление *переменной* выполняется командой `DECLARE` , имеющей следующий формат:

```
DECLARE {@имя_переменной тип_данных }
        [,...n]
```

Значения *переменной* можно присвоить посредством команд `SET` и `SELECT` . С помощью команды `SELECT` *переменной* можно присвоить не только конкретное *значение*, но и результат вычисления *выражения*.

```
DECLARE @a INT
SET @a=10

2.4. Использование SET для присваивания значения локальной переменной.

DECLARE @k INT
SELECT @k=SUM(количество) FROM Товар

2.5. Использование SELECT для присваивания локальной переменной результата вычислений.
```

Управляющие конструкции SQL

Язык *SQL* является непроцедурным, но тем не менее в среде *SQL Server* предусмотрен ряд различных *управляющих конструкций*, без которых невозможно написание эффективных алгоритмов.

Группировка двух и более команд в единый *блок* осуществляется с использованием ключевых слов `BEGIN` и `END` :

```
<блок_операторов>::=
BEGIN
{ sql_оператор | блок_операторов }
END
```

Сгруппированные команды воспринимаются интерпретатором *SQL* как одна *команда*. Подобная группировка требуется для *конструкций поливариантных ветвлений* , *условных* и *циклических* конструкций. *Блоки* `BEGIN...END` могут быть вложенными.

Некоторые команды *SQL* не должны выполняться вместе с другими командами (речь идет о командах резервного копирования, изменения структуры таблиц, хранимых процедур и им подобных), поэтому их совместное включение в конструкцию `BEGIN...END` не допускается.

Нередко определенная часть программы должна выполняться только при реализации некоторого логического условия. *Синтаксис условного оператора* показан ниже:

```
<условный_оператор>::=
IF лог_выражение
  { sql_оператор | блок_операторов }
[ ELSE
  {sql_оператор | блок_операторов } ]
```

Циклы организуются с помощью следующей конструкции:

```
<оператор_цикла>::=
WHILE лог_выражение
   { sql_оператор | блок_операторов }
   [ BREAK ]
   { sql_оператор | блок_операторов }
   [ CONTINUE ]
```

Цикл можно принудительно остановить, если в его теле выполнить команду `BREAK` . Если же нужно начать цикл заново, не дожидаясь выполнения всех команд в теле, необходимо выполнить команду `CONTINUE` .

Для замены *множества* одиночных или вложенных *условных операторов* используется следующая конструкция:

```
<оператор_поливариантных_ветвлений>::=
CASE входное_значение
WHEN {значение_для_сравнения |
   лог_выражение } THEN
        вых_выражение [,...n]
[ ELSE иначе_вых_выражение ]
END
```

Если входное *значение* и *значение* для сравнения совпадают, то конструкция возвращает выходное *значение*. Если же *значение* входного параметра не найдено ни в одной из строк `WHEN...THEN` , то тогда будет возвращено *значение*, указанное после ключевого слова `ELSE` .

Основные объекты структуры базы данных SQL-сервера

Рассмотрим логическую структуру базы *данных*.

Логическая структура определяет структуру таблиц, взаимоотношения между ними, *список* пользователей, хранимые процедуры, правила, умолчания и другие *объекты базы данных*.

Логически *данные* в *SQL Server* организованы в виде *объектов*. К основным *объектам базы данных SQL Server* относятся *объекты*, представленные в таблице 2.4.

Таблица 2.4.

`Tables`	Таблицы базы *данных*, в которых хранятся собственно данные

Views	Просмотры (виртуальные таблицы) для отображения *данных* из таблиц
Stored Procedures	Хранимые процедуры
Triggers	Триггеры – специальные хранимые процедуры, вызываемые при изменении *данных* в таблице
User Defined function	Создаваемые пользователем функции
Indexes	Индексы – дополнительные структуры, призванные повысить производительность работы с *данными*
User Defined Data Types	Определяемые пользователем *типы данных*
Keys	Ключи – один из видов *ограничений целостности данных*
Constraints	Ограничение целостности – *объекты* для обеспечения логической целостности данных
Users	Пользователи, обладающие доступом к базе данных
Roles	Роли, позволяющие объединять пользователей в группы
Rules	Правила базы *данных*, позволяющие контролировать логическую целостность данных
Defaults	Умолчания или стандартные установки базы данных

Приведем краткий обзор основных *объектов* баз *данных*.

Таблицы

Все *данные* в SQL содержатся в *объектах*, называемых таблицами. Таблицы представляют собой совокупность каких-либо сведений об *объектах*, явлениях, процессах реального мира. Никакие другие *объекты* не хранят *данные*, но они могут обращаться к *данным* в таблице. Таблицы в SQL имеют такую же структуру, что и таблицы всех других СУБД и содержат:

- строки; каждая строка (или запись) представляет собой совокупность атрибутов (свойств) конкретного экземпляра *объекта* ;
- столбцы; каждый столбец (поле) представляет собой атрибут или совокупность атрибутов. Поле строки является минимальным элементом таблицы. Каждый столбец в таблице имеет определенное имя, *тип данных* и размер.

Представления

Представлениями (просмотрами) называют виртуальные таблицы, содержимое которых определяется запросом. Подобно *реальным таблицам*, представления содержат именованные столбцы и строки с *данными*. Для конечных пользователей представление выглядит как таблица, но в действительности оно не содержит *данных*, а лишь представляет *данные*, расположенные в одной или нескольких таблицах. Информация, которую видит пользователь через представление, не сохраняется в базе *данных* как самостоятельный *объект*.

Хранимые процедуры

Хранимые процедуры представляют собой группу команд SQL, объединенных в один модуль. Такая группа команд компилируется и выполняется как единое целое.

Триггеры

Триггерами называется специальный класс хранимых процедур, автоматически запускаемых при добавлении, изменении или удалении *данных* из таблицы.

Функции

Функции в языках программирования – это конструкции, содержащие часто исполняемый код. Функция выполняет какие-либо действия над *данными* и возвращает некоторое значение.

Индексы

Индекс – структура, связанная с таблицей или представлением и предназначенная для ускорения поиска информации в них. Индекс определяется для одного или нескольких столбцов, называемых индексированными столбцами. Он содержит отсортированные значения индексированного столбца или столбцов со ссылками на соответствующую строку исходной таблицы или представления. Повышение

производительности достигается за счет сортировки *данных*. Использование индексов может существенно повысить производительность поиска, однако для хранения индексов необходимо дополнительное пространство в базе *данных*.

Пользовательские типы данных

Пользовательские типы данных – это *типы данных*, которые создает пользователь на основе системных *типов данных*, когда в нескольких таблицах необходимо хранить однотипные значения; причем нужно гарантировать, что столбцы в таблице будут иметь одинаковый размер, *тип данных* и чувствительность к значениям `NULL` .

Ограничения целостности

Ограничения целостности – механизм, обеспечивающий автоматический контроль соответствия *данных* установленным условиям (или ограничениям). Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. К ограничениям целостности относятся: ограничение на значение `NULL` , проверочные ограничения, ограничение уникальности (уникальный ключ), ограничение первичного ключа и ограничение внешнего ключа. Последние три ограничения тесно связаны с понятием ключей.

Правила

Правила используются для ограничения значений, хранимых в столбце таблицы или в *пользовательском типе данных*. Они существуют как самостоятельные *объекты* базы *данных*, которые связываются со столбцами таблиц и *пользовательскими типами данных* . Контроль значений *данных* может быть реализован и с помощью ограничений целостности.

Умолчания

Умолчания – самостоятельный *объект базы данных*, представляющий значение, которое будет присвоено элементу таблицы при вставке строки, если в команде вставки явно не указано значение для этого столбца.