

***Лекция 10. Диалекты языка SQL в СУБД. Встроенные функции, переменные и управляющие конструкции***

**Виды диалектов языка SQL. Основные понятия языка T-SQL. Соглашения о синтаксисе T-SQL. Многокомпонентные имена. Идентификатор. Константы. Типы данных. Локальные, глобальные переменные. Объявление и инициализация переменной Переменные в запросах. Программирование в T-SQL. Пакеты. Команда GO.**

**Встроенные функции, переменные и управляющие конструкции в T-SQL. Управляющая конструкции BEGIN...END. Условная конструкция IF...ELSE. Цикл WHILE. Операторы BREAK, CONTINUE, END. Обработка ошибок. Функции для работы со строками. Функции для работы с числами. Функции по работе с датами и временем. Преобразование данных (функции CONVERT и CAST). Оценка списка условий и возвращение одного из нескольких возможных выражений результатов (выражение CASE). Логические функции – IF. Функции NEWID, ISNULL и COALESCE.**

10.1 Виды диалектов языка SQL. Основные понятия языка T-SQL. Соглашения о синтаксисе T-SQL. Многокомпонентные имена. Идентификатор. Константы. Типы данных. Локальные, глобальные переменные. Объявление и инициализация переменной Переменные в запросах. Программирование в T-SQL. Пакеты. Команда GO.

*Рассмотрим концепцию программируемых объектов и познакомимся с возможностями, которые Microsoft SQL Server предоставляет в этой области.*

*Что такое переменные, пакеты, инструкции для управления потоком выполнения, **курсоры**, временные таблицы, процедуры (пользовательские функции, триггеры и хранимые процедуры) и динамические элементы языка SQL.*

### 10.1.1 Переменные

**1) Переменные** предназначены для временного хранения данных и их дальнейшего использования в рамках одного и того же пакета.

**Пакет** это одна или несколько команд, которые передаются в SQL Server и выполняются как единое целое.

Для объявления переменных используется команда DECLARE; с помощью команды SET происходит присваивание значений.

Упрощенный синтаксис команды имеет следующий вид:

**DECLARE** <@название> AS <тип>

Имена переменных в Transact-SQL начинаются с символа @.

Объявить сразу несколько переменных одним оператором DECLARE можно так:

**DECLARE** <@название1> AS <тип1>, ..., <@названиеN> AS <типN>

Ключевое слово AS необязательно.

При объявлении переменной можно ее инициализировать:

**DECLARE** <@название> AS <тип> = <значение>

Например, следующий код объявляет переменную под названием @i типа INT и присваивает ей число 10.

**DECLARE** @i AS INT;

**SET** @i = 10;

Объявление и инициализация переменных в рамках одной команды, как показано ниже.

**DECLARE** @i AS INT = 10;

**Значение, которое присваивается скалярной переменной, должно быть результатом скалярного выражения (даже если речь идет о вложенном запросе).**

Например, следующий код объявляет переменную под названием *@empname* и присваивает ей результат выполнения вложенного скалярного запроса, который *возвращает полное имя сотрудника с идентификатором 3*.

***USE TSQL2012;***

***DECLARE @empname AS NVARCHAR(31);***

***SET @empname = (SELECT firstname + N' ' + lastname FROM  
HR.Employees***

***WHERE empid = 3);***

***SELECT @empname AS empname;***

### ***Вывод: Переменные в T-Sql***

Переменная представляет именованный объект, который хранит некоторое значение. Для определения переменных применяется выражение **DECLARE**, после которого указывается название и тип переменной. При этом название локальной переменной должно начинаться с символа @:

**DECLARE @название\_переменной тип\_данных**

Например, определим переменную name, которая будет иметь тип NVARCHAR:

**DECLARE @name NVARCHAR(20)**

Также можно определить через запятую сразу несколько переменных:

**DECLARE @name NVARCHAR(20), @age INT**

С помощью выражения SET можно присвоить переменной некоторое значение:

**DECLARE @name NVARCHAR(20), @age INT;**

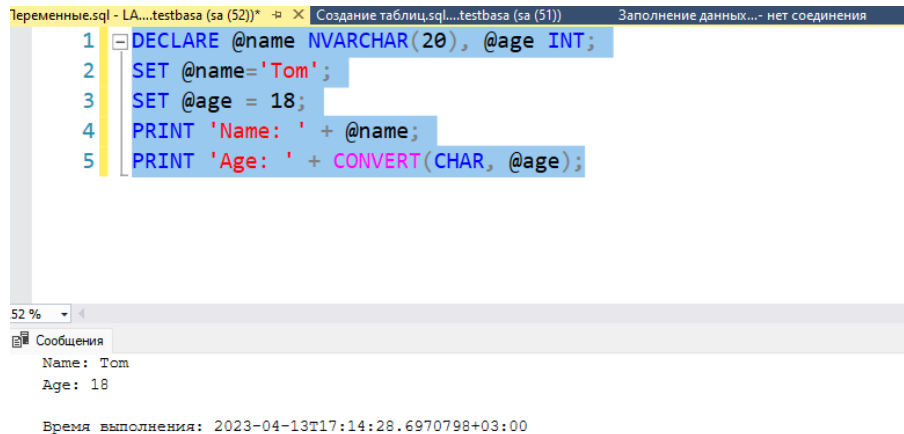
**SET @name='Tom';**

**SET @age = 18;**

Так как @name предоставляет тип NVARCHAR, то есть строку, то этой переменной соответственно и присваивается строка. А переменной @age присваивается число, так как она представляет тип INT.

**Выражение PRINT возвращает сообщение клиенту.**

Например:



```
1 DECLARE @name NVARCHAR(20), @age INT;  
2 SET @name='Tom';  
3 SET @age = 18;  
4 PRINT 'Name: ' + @name;  
5 PRINT 'Age: ' + CONVERT(CHAR, @age);
```

Сообщения

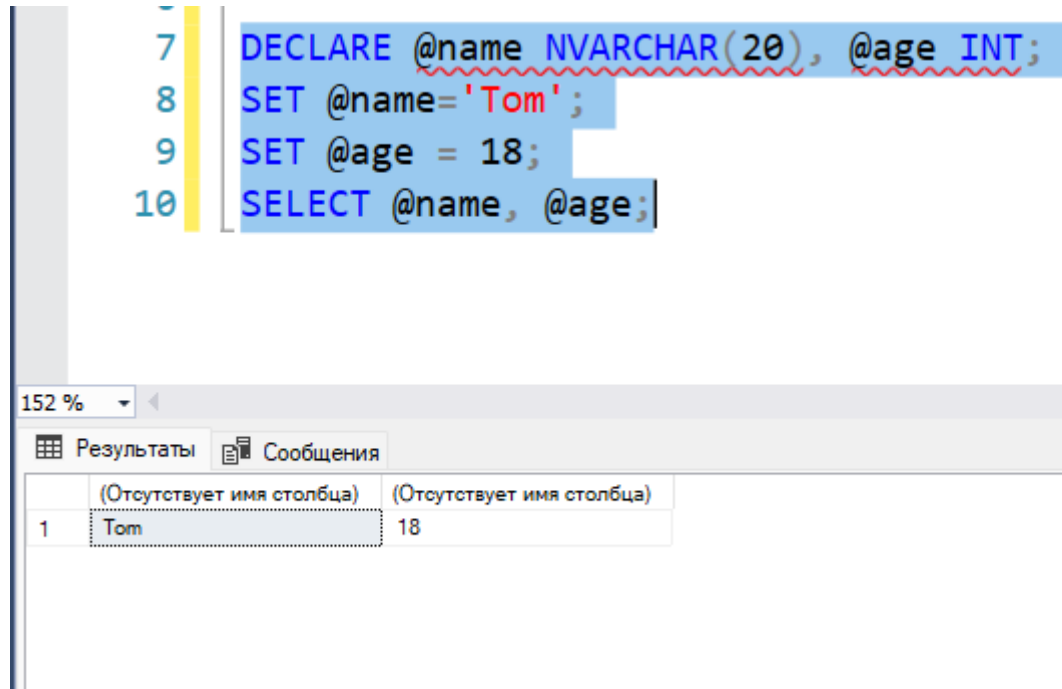
Name: Tom  
Age: 18

Время выполнения: 2023-04-13T17:14:28.6970798+03:00

**Также можно использовать для получения значения команду SELECT:**

Значения переменных можно вывести с помощью команды SELECT. Синтаксис команды имеет следующий вид:

*SELECT <@переменная1> [AS псевдоним1], ..., <@переменнаяN> [AS псевдонимN]*



```
7 DECLARE @name NVARCHAR(20), @age INT;  
8 SET @name='Tom';  
9 SET @age = 18;  
10 SELECT @name, @age;
```

152 %

Результаты   Сообщения

	(Отсутствует имя столбца)	(Отсутствует имя столбца)
1	Tom	18

*Команда SET может одновременно работать только с одной переменной, поэтому чтобы присвоить значения нескольким атрибутам, вам понадобится соответствующее количество таких команд.*

*В связи с этим получение содержимого нескольких атрибутов одной и той же строки может привести к избыточности вашего кода.*

*В примере, представленном ниже, используются две отдельные команды SET, которые присваивают переменным имя и фамилию сотрудника с идентификатором 3.*

```
DECLARE @firstname AS NVARCHAR(10),  
@lastname AS NVARCHAR(20);  
SET @firstname = (SELECT firstname  
FROM HR.Employees  
WHERE empid = 3);  
SET @lastname = (SELECT lastname  
FROM HR.Employees WHERE empid = 3);  
SELECT @firstname AS firstname, @lastname AS lastname;
```

## 2) Переменные в запросах

Через переменные мы можем передавать данные в запросы. И также мы можем получать данные, которые являются результатом запросов, в переменные.

Например, при выборке из таблиц с помощью команды *SELECT* мы можем извлекать данные в переменную с помощью следующего синтаксиса:

### ***SELECT***

***@переменная\_1 = спецификация\_столбца\_1,***

***@переменная\_2 = спецификация\_столбца\_2,***

***@переменная\_N = спецификация\_столбца\_N***

**Кроме того, в выражении *SET* значение, присваиваемое переменной, также может быть результатом команды *SELECT*.**

Пусть Например, пусть у нас будут следующие таблицы:

```
3 CREATE TABLE Products
4 (
5     Id INT IDENTITY PRIMARY KEY,
6     ProductName NVARCHAR(30) NOT NULL,
7     Manufacturer NVARCHAR(20) NOT NULL,
8     ProductCount INT DEFAULT 0,
9     Price MONEY NOT NULL
10 );
11 CREATE TABLE Customers
12 (
13     Id INT IDENTITY PRIMARY KEY,
14     FirstName NVARCHAR(30) NOT NULL
15 );
16 CREATE TABLE Orders
17 (
18     Id INT IDENTITY PRIMARY KEY,
19     --ON DELETE CASCADE используется для автоматического удаления строк
20     --из дочерней таблицы при удалении строк из родительской таблицы
21     ProductId INT NOT NULL REFERENCES Products(Id) ON DELETE CASCADE,
22     CustomerId INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,
23     CreatedAt DATE NOT NULL,
24     ProductCount INT DEFAULT 1,
25     Price MONEY NOT NULL
26 );
```



*Используем переменные при извлечении данных:*

```
--
12  --Используем переменные при извлечении данных
13  DECLARE
14      @maxPrice MONEY,
15      @minPrice MONEY,
16      @dif MONEY,
17      @count INT
18
19  SET @count = (SELECT SUM(ProductCount) FROM Orders);
20
21  SELECT @minPrice=MIN(Price), @maxPrice = MAX(Price) FROM Products
22
23  SET @dif = @maxPrice - @minPrice;
24
25  PRINT 'Всего продано: ' + STR(@count, 5) + ' товара(ов)';
26  PRINT 'Разница между максимальной и минимальной ценой: ' + STR(@dif)
27
```

152 %

Сообщения

Всего продано: 4 товара(ов)  
Разница между максимальной и минимальной ценой: 30000

Время выполнения: 2023-04-13T21:57:52.2134371+03:00

*В данном случае переменная @count будет содержать сумму всех значений из столбца ProductCount таблицы Orders, то есть общее количество проданных товаров.*

*Переменные @min и @max хранят соответственно минимальное и максимальное значения столбца Price из таблицы Products, а переменная @dif - разницу между этими значениями. И подобно простым значениям, переменные также могут участвовать в операциях.*

## Другой пример

Здесь извлекаемые данные из двух таблиц *Products* и *Orders* группируются по столбцам *Id* и *ProductName* из таблицы *Products*. Затем данные фильтруются по столбцу *Id* из *Products*. А извлеченные данные попадают в переменные *@sum*, *@name*, *@prodid*.

```
27
28 --Здесь извлекаемые данные из двух таблиц Products и Orders
29 --группируются по столбцам Id и ProductName из таблицы Products.
30 --Затем данные фильтруются по столбцу Id из Products.
31 --А извлеченные данные попадают в переменные @sum, @name, @prodid.
32
33 DECLARE @sum MONEY, @id INT, @prodid INT, @name NVARCHAR(20);
34 SET @id=2;
35
36 SELECT @sum = SUM(Orders.Price*Orders.ProductCount),
37         @name=Products.ProductName, @prodid = Products.Id
38 FROM Orders
39 INNER JOIN Products ON ProductId = Products.Id
40 GROUP BY Products.ProductName, Products.Id
41 HAVING Products.Id=@id
42
43 PRINT 'Товар ' + @name + ' продан на сумму ' + STR(@sum)
44
45
```

152 %

Сообщения

Товар iPhone 6S продан на сумму 82000

Время выполнения: 2023-04-14T05:11:09.7382686+03:00

*FROM [testbasa].[dbo].[Orders]*

152 %

Результаты

	Id	ProductId	CustomerId	CreatedAt	ProductCount	Price
1	1	4	1	2017-07-11	2	46000,00
2	2	2	1	2017-07-13	1	41000,00
3	3	2	2	2017-07-11	1	41000,00

*FROM [testbasa].[dbo].[Products]*

152 %

Результаты

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	2	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi 5X	Xiaomi	2	26000,00
7	7	OnePlus 5	OnePlus	6	38000,00

*Управляющие конструкции в T-SQL. Управляющая конструкции BEGIN...END. Условная конструкция IF...ELSE. Цикл WHILE. Операторы BREAK, CONTINUE, END. Обработка ошибок.*

### 3) Условные выражения

Для выполнения действий по условию используется выражение IF ... ELSE.

SQL Server вычисляет выражение после ключевого слова IF.

И если оно **истинно**, то выполняются инструкции после ключевого слова IF.

Если условие **ложно**, то выполняются инструкции после ключевого слова ELSE.

Если после IF или ELSE располагается блок инструкций, то этот блок заключается между ключевыми словами **BEGIN и END:**

**IF условие**

**{инструкция|BEGIN...END}**

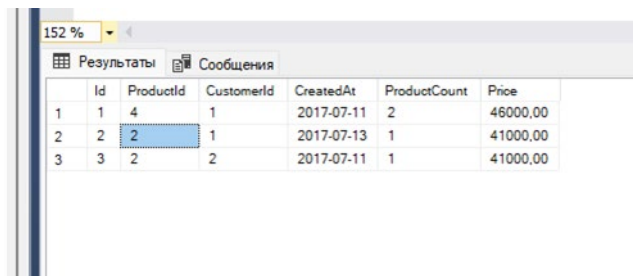
**[ELSE**

**{инструкция|BEGIN...END}]**

Выражение ELSE является **необязательным**, и его можно опускать.

## Пример:

Таблица **Orders** представляет заказы, а столбец **CreatedAt** - дату заказов. Узнаем, были ли заказы за последние 10 дней:



	Id	ProductId	CustomerId	CreatedAt	ProductCount	Price
1	1	4	1	2017-07-11	2	46000,00
2	2	2	1	2017-07-13	1	41000,00
3	3	2	2	2017-07-11	1	41000,00

**GETDATE**: возвращает текущую локальную дату и время на основе системных часов в виде объекта *datetime*

**SELECT GETDATE()**

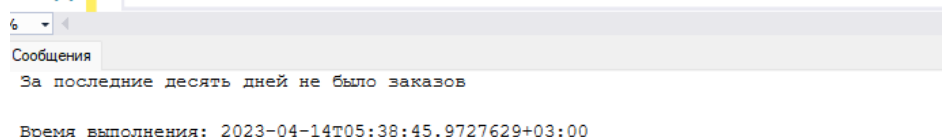
**DAY**: возвращает день даты, который передается в качестве параметра

**SELECT DAY(GETDATE())**

**DATEDIFF**: возвращает разницу между двумя датами. Первый параметр - компонент даты, который указывает, в каких единицах стоит измерять разницу. Второй и третий параметры - сравниваемые даты

**SELECT DATEDIFF(day, '2017-7-28', '2018-9-28')**

```
68  --заказы за последние 10 дней
69  DECLARE @lastDate DATE
70
71  SELECT @lastDate = MAX(CreatedAt) FROM Orders
72
73  IF DATEDIFF(day, @lastDate, GETDATE()) > 10
74      PRINT 'За последние десять дней не было заказов'
75
76
77
```



Сообщения

За последние десять дней не было заказов

Время выполнения: 2023-04-14T05:38:45.9727629+03:00

## Добавим выражение *ELSE*:

```
77 --Добавим выражение ELSE
78
79 DECLARE @lastDate DATE
80
81 SELECT @lastDate = MAX(CreatedAt) FROM Orders
82
83 IF DATEDIFF(day, @lastDate, GETDATE()) > 10
84     PRINT 'За последние десять дней не было заказов'
85 ELSE
86     PRINT 'За последние десять дней были заказы'
87
```

152 %  
Сообщения  
За последние десять дней не было заказов  
Время выполнения: 2023-04-14T05:44:22.9571554+03:00

*Если после IF или ELSE идут две и более инструкций, то они заключаются в блок BEGIN...END:*

```
90 DECLARE @lastDate DATE, @count INT, @sum MONEY
91
92 SELECT @lastDate = MAX(CreatedAt),
93        @count = SUM(ProductCount),
94        @sum = SUM(ProductCount * Price)
95 FROM Orders
96
97 IF @count > 0
98 BEGIN
99     PRINT 'Дата последнего заказа: ' + CONVERT(NVARCHAR, @lastDate)
100    PRINT 'Продано ' + CONVERT(NVARCHAR, @count) + ' единиц(ы)'
101    PRINT 'На общую сумму ' + CONVERT(NVARCHAR, @sum)
102 END
103 ELSE
104     PRINT 'Заказы в базе данных отсутствуют'
```

152 %  
Сообщения  
Дата последнего заказа: 2017-07-13  
Продано 4 единиц(ы)  
На общую сумму 174000.00  
Время выполнения: 2023-04-14T05:46:35.1765491+03:00

*В тех случаях, когда необходимо выполнить преобразования от типов с высшим приоритетом к типам с низшим приоритетом, то надо выполнять явное приведение типов. Для этого в T-SQL определены две функции: CONVERT и CAST.*

**CONVERT(тип\_данных, выражение [, стиль])**

*Третий необязательный параметр задает стиль форматирования данных. Этот параметр представляет числовое значение, которое для разных типов данных имеет разную интерпретацию. Например, некоторые значения для форматирования дат и времени:*

*0 или 100 - формат даты "Mon dd уууу hh:miAM/PM" (значение по умолчанию)*

*1 или 101 - формат даты "mm/dd/уууу"*

*3 или 103 - формат даты "dd/mm/уууу"*

*7 или 107 - формат даты "Mon dd, уууу hh:miAM/PM"*

*8 или 108 - формат даты "hh:mi:ss"*

*10 или 110 - формат даты "mm-dd-уууу"*

*14 или 114 - формат даты "hh:mi:ss:тттт" (24-часовой формат времени)*

*Некоторые значения для форматирования данных типа money в строку:*

*0 - в дробной части числа остаются только две цифры (по умолчанию)*

*1 - в дробной части числа остаются только две цифры, а для разделения разрядов применяется запятая*

*2 - в дробной части числа остаются только четыре цифры*

#### 4) Циклы

Для выполнения повторяющихся операций в T-SQL применяются циклы. В частности, в T-SQL есть цикл **WHILE**.

Этот цикл выполняет определенные действия, **пока некоторое условие истинно**.

**WHILE** условие

{инструкция|**BEGIN...END**}

Например, **вычислим факториал числа**:

```
105
106      --вычислим факториал числа
107      DECLARE @number INT, @factorial INT
108      SET @factorial = 1;
109      SET @number = 5;
110
111      WHILE @number > 0
112      BEGIN
113          SET @factorial = @factorial * @number
114          SET @number = @number - 1
115      END;
116
```

Сообщения  
120

Время выполнения: 2023-04-14T05:57:09.4041554+03:00

То есть в данном случае пока переменная **@number** не будет равна 0, будет продолжаться цикл **WHILE**. Так как **@number** равна 5, то цикл сделает пять проходов. Каждый проход цикла называется итерацией. В каждой итерации будет переуставляться значение переменных **@factorial** и **@number**.

## *Другой пример - рассчитаем баланс счета через несколько лет с учетом процентной ставки*

```
119  --Другой пример - рассчитаем баланс счета через несколько лет с учетом процентной ставки:
120
121  CREATE TABLE #Accounts1 ( CreatedAt DATE, Balance MONEY)
122
123  DECLARE @rate FLOAT, @period INT, @sum MONEY, @date DATE
124  SET @date = GETDATE()
125  SET @rate = 0.065;
126  SET @period = 5;
127  SET @sum = 10000;
128
129  WHILE @period > 0
130  BEGIN
131      INSERT INTO #Accounts1 VALUES(@date, @sum)
132      SET @period = @period - 1
133      SET @date = DATEADD(year, 1, @date)
134      SET @sum = @sum + @sum * @rate
135  END;
136
137  SELECT * FROM #Accounts1
138
139
140
```

Результаты		Сообщения
CreatedAt	Balance	
2023-04-14	10000,00	
2024-04-14	10650,00	
2025-04-14	11342,25	
2026-04-14	12079,4963	
2027-04-14	12864,6636	

*Здесь создается временная таблица #Accounts, в которую добавляется в цикле пять строк с данными.*

**DATEADD:** возвращает дату, которая является результатом сложения числа к определенному компоненту даты.

*Первый параметр представляет компонент даты, для функции DATENAME.*

*Второй параметр - добавляемое количество.*

*Третий параметр - сама дата, к которой надо сделать прибавление:*

**SELECT DATEADD(month, 2, '2017-7-28')**

**SELECT DATEADD(day, 5, '2017-7-28')**

**SELECT DATEADD(day, -5, '2017-7-28')**

**DATENAME:** возвращает часть даты в виде строки. Параметр выбора части даты передается в качестве первого параметра, а сама дата передается в качестве второго параметра:

**SELECT DATENAME(month, GETDATE())**

### **Операторы BREAK и CONTINUE**

Оператор **BREAK** позволяет завершить цикл, а оператор **CONTINUE** - перейти к новой итерации.

```
139 --операторы BREAK и CONTINUE
140 DECLARE @number INT
141 SET @number = 1
142
143 WHILE @number < 10
144 BEGIN
145     PRINT CONVERT(NVARCHAR, @number)
146     SET @number = @number + 1
147     IF @number = 7
148         BREAK;
149     IF @number = 4
150         CONTINUE;
151     PRINT 'Конец итерации'
152 END;
153
154
```

152 %

Сообщения

```
1
Конец итерации
2
Конец итерации
3
4
Конец итерации
5
Конец итерации
6
```

Когда переменная `@number` станет равна 4, то с помощью оператора **CONTINUE** произойдет переход к новой итерации, поэтому последующая строка `PRINT 'Конец итерации'` не будет выполняться, хотя цикл продолжится.

Когда переменная `@number` станет равна 7, то оператор **BREAK** произведет выход из цикла, и он завершится.



### 5) Обработка ошибок

Для обработки ошибок в T-SQL применяется конструкция **TRY...CATCH**. Она имеет следующий формальный синтаксис:

```
BEGIN TRY  
    инструкции  
END TRY  
BEGIN CATCH  
    инструкции  
END CATCH
```

Между выражениями **BEGIN TRY** и **END TRY** помещаются инструкции, которые потенциально могут вызвать ошибку, например, какой-нибудь запрос.

И если в этом блоке **TRY** возникнет ошибка, то управление передается в блок **CATCH**, где можно обработать ошибку.

В блоке **CATCH** для обработки ошибки мы можем использовать ряд функций:

**ERROR\_NUMBER():** возвращает номер ошибки

**ERROR\_MESSAGE():** возвращает сообщение об ошибке

**ERROR\_SEVERITY():** возвращает степень серьезности ошибки.

Степень серьезности представляет числовое значение. *И если оно равно 10 и меньше, то такая ошибка рассматривается как предупреждение и не обрабатывается конструкцией TRY...CATCH.*

Если же это значение **равно 20** и выше, то такая ошибка приводит к закрытию подключения к базе данных, если она не обрабатывается конструкцией TRY...CATCH.

**ERROR\_STATE():** возвращает состояние ошибки.

Например, добавим в таблицу данные, которые не соответствуют ограничениям столбцов:

```
156  
157 CREATE TABLE Accounts (FirstName NVARCHAR NOT NULL, Age INT NOT NULL)  
158  
159 BEGIN TRY  
160     INSERT INTO Accounts VALUES(NULL, NULL)  
161     PRINT 'Данные успешно добавлены!'  
162 END TRY  
163 BEGIN CATCH  
164     PRINT 'Error ' + CONVERT(VARCHAR, ERROR_NUMBER()) + ':' + ERROR_MESSAGE()  
165 END CATCH  
166
```

152 %  
Сообщения

(загружено строк: 0)  
Error 515: Cannot insert the value NULL into column 'FirstName', table 'testbasa.dbo.Accounts'; column does not allow nulls. INSERT fails.

**Для работы со строками в T-SQL можно применять следующие функции:**

**Список часто используемых *строковых* функций:**

<b>LEN(строка)</b>	возвращает количество символов в заданной строке
<b>TRIM(строка)</b> <b>TRIM([символ FROM] строка)</b>	удаляет символ пробела или другие заданные символы в начале и в конце строки.
<b>LTRIM(строка)</b>	удаляет начальные пробелы из заданной строки
<b>RTRIM(строка)</b>	удаляет конечные пробелы из заданной строки
<b>CHARINDEX(подстрока, строка)</b> <b>CHARINDEX(подстрока, строка, начальная позиция)</b>	возвращает индекс, по которому находится первое вхождение подстроки в строке.
<b>PATINDEX('%шаблон%', строка)</b>	возвращает индекс, по которому находится первое вхождение определенного шаблона в строке
<b>LEFT(строка, число)</b>	возвращает с начала строки определенное количество символов
<b>RIGHT(строка, число)</b>	возвращает с конца строки определенное количество символов
<b>SUBSTRING(строка, начальная позиция, длина )</b>	возвращает подстроку заданной длиной, начиная с данной позиции
<b>REPLACE(строка, подстрока, замена)</b>	заменяет одну подстроку другой
<b>REVERSE(строка)</b>	переворачивает строку наоборот
<b>CONCAT(строка1, строка2 [, строкаN ] )</b>	объединяет заданные строки в одну
<b>LOWER(строка)</b>	переводит строку в нижний регистр
<b>UPPER (строка)</b>	переводит строку в верхний регистр
<b>SPACE(число)</b>	возвращает заданное количество пробелов
<b>REPLICATE(строка, число)</b>	повторяет значение строки указанное число раз
<b>STUFF(строка, начальная позиция, количество, замена)</b>	удаляет указанное количество символов первой строки в начальной позиции и вставляет на их место замену.

```

167 --LEN: возвращает количество символов в строке.
168 --В качестве параметра в функцию передается строка, для которой надо найти длину:
169 SELECT LEN('Apple')
170

```

Результаты Сообщения

	(Отсутствует имя столбца)
5	

```

171 --LTRIM: удаляет начальные пробелы из строки.
172 --В качестве параметра принимает строку:
173 SELECT LTRIM(' Apple')
174
175 --RTRIM: удаляет конечные пробелы из строки.
176 --В качестве параметра принимает строку:
177 SELECT RTRIM(' Apple ')
178

```

152 %

Результаты Сообщения

	(Отсутствует имя столбца)
1	Apple
	(Отсутствует имя столбца)
1	Apple

```

179 --CHARINDEX: возвращает индекс, по которому находится первое вхождение подстроки в строке.
180 --В качестве первого параметра передается подстрока, а в качестве второго - строка, в которой надо вести поиск:
181 SELECT CHARINDEX('pl', 'Apple')
182
183 --PATINDEX: возвращает индекс, по которому находится первое вхождение определенного шаблона в строке:
184 SELECT PATINDEX('%p_e%', 'Apple')
185

```

52 %

Результаты Сообщения

	(Отсутствует имя столбца)
1	3

	(Отсутствует имя столбца)
1	3

```

186 --LEFT: вырезает с начала строки определенное количество символов.
187 --Первый параметр функции - строка,
188 --а второй - количество символов, которые надо вырезать сначала строки:
189 SELECT LEFT('Apple', 3) -- App
190
191 --RIGHT: вырезает с конца строки определенное количество символов.
192 --Первый параметр функции - строка,
193 --а второй - количество символов, которые надо вырезать сначала строки:
194 SELECT RIGHT('Apple', 3)

```

152 %

Результаты Сообщения

	(Отсутствует имя столбца)
1	App

	(Отсутствует имя столбца)
1	ple

```
195
196 --SUBSTRING: вырезает из строки подстроку определенной длиной,
197 --начиная с определенного индекса. Первый параметр функции - строка,
198 --второй - начальный индекс для вырезки, и третий параметр - количество вырезаемых символов:
199 SELECT SUBSTRING('Galaxy S8 Plus', 8, 2)
200
201 --REPLACE: заменяет одну подстроку другой в рамках строки.
202 --Первый параметр функции - строка,
203 --второй - подстрока, которую надо заменить,
204 --а третий - подстрока, на которую надо заменить:
205 SELECT REPLACE('Galaxy S8 Plus', 'S8 Plus', 'Note 8')
206
```

152 %

Результаты Сообщения

	(Отсутствует имя столбца)
1	S8

	(Отсутствует имя столбца)
1	Galaxy Note 8

```
207 --REVERSE: переворачивает строку наоборот:
208 SELECT REVERSE('123456789')
209
210 --CONCAT: объединяет две строки в одну.
211 --В качестве параметра принимает от 2-х и более строк, которые надо соединить:
212 SELECT CONCAT('Tom', ' ', 'Smith')
213
214 --LOWER: переводит строку в нижний регистр
215 SELECT LOWER('Apple')
216
217 --UPPER: переводит строку в верхний регистр
218 SELECT UPPER('Apple') -- APPLE
```

152 %

Результаты Сообщения

	(Отсутствует имя столбца)
1	987654321

	(Отсутствует имя столбца)
1	Tom Smith

	(Отсутствует имя столбца)
1	apple

	(Отсутствует имя столбца)
1	APPLE

SPACE: возвращает строку, которая содержит определенное количество пробелов

## Пример

152 %

Результаты Сообщения

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	2	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi 5X	Xiaomi	2	26000,00
7	7	OnePlus 5	OnePlus	6	38000,00

```
219
220 --пример
221 SELECT UPPER(LEFT(Manufacturer,2)) AS Abbreviation,
222        CONCAT(ProductName, ' - ', Manufacturer) AS FullProdName
223 FROM Products
224 ORDER BY Abbreviation
225
```

152 %

Результаты Сообщения

	Abbreviation	FullProdName
1	AP	iPhone 6 - Apple
2	AP	iPhone 6S - Apple
3	AP	iPhone 7 - Apple
4	ON	OnePlus 5 - OnePlus
5	SA	Galaxy S8 - Samsung
6	SA	Galaxy S8 Plus - Samsung
7	XI	Mi 5X - Xiaomi

## Функции для работы с числами

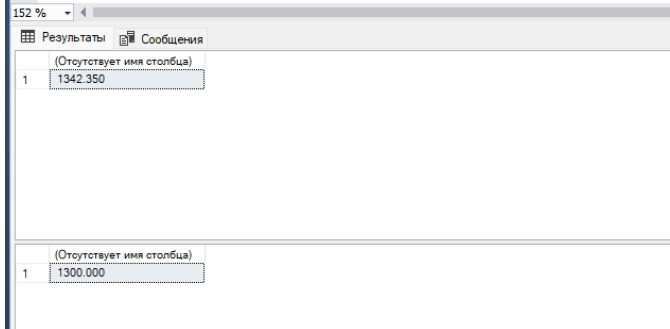
### Список часто используемых числовых функций:

<b>ABS(число)</b>	возвращает абсолютное значение числа
<b>CEILING(число)</b>	возвращает наименьшее целое, большее или равное заданного числа.
<b>FLOOR(число)</b>	возвращает наибольшее целое число, меньшее или равное заданного числа
<b>POWER(число, степень)</b>	возвращает значение указанного выражения, возведенное в заданную степень
<b>RAND([начальное значение])</b>	возвращает псевдослучайное значение от 0 до 1
<b>ROUND(число, точность)</b>	возвращает число, округленное до указанной точности
<b>SIGN(число)</b>	возвращает положительное (+1), нулевое (0) или отрицательное (-1) значение, обозначающее знак заданного выражения
<b>SQRT(число)</b>	возвращает квадратный корень данного числа
<b>SQUARE(число)</b>	возвращает квадрат указанного числа
<b>PI()</b>	возвращает константное значение $\pi$
<b>ACOS(число)</b>	возвращает угол в радианах, косинус которого задан – арккосинус.
<b>ASIN(число)</b>	возвращает угол в радианах, синус которого задан – арксинус.
<b>ATAN(число)</b>	возвращает угол в радианах, тангенс которого задан – арктангенс.
<b>COS(число)</b>	возвращает косинус указанного угла в радианах.
<b>SIN(число)</b>	возвращает синус указанного угла в радианах.
<b>TAN(число)</b>	возвращает тангенс указанного угла в радианах.
<b>COT(число)</b>	возвращает котангенс указанного угла в радианах
<b>DEGREES(число)</b>	возвращает для значения угла в радианах соответствующее значение в градусах.
<b>RADIANS(число)</b>	возвращает для значения угла в градусах соответствующее значение в радианах
<b>EXP(число)</b>	возвращает экспонент заданного числа
<b>LOG(число)</b>	возвращает натуральный логарифм указанного числа
<b>LOG(число, основа)</b>	возвращает логарифм указанного числа
<b>LOG10(число)</b>	возвращает десятичный логарифм указанного числа

## Примеры

**ROUND:** округляет число. В качестве первого параметра передается число. Второй параметр указывает на длину. Если длина представляет положительное число, то оно указывает, до какой цифры после запятой идет округление. Если длина представляет отрицательное число, то оно указывает, до какой цифры с конца числа до запятой идет округление

```
226 --ROUND
227 SELECT ROUND(1342.345, 2) -- 1342.350
228 SELECT ROUND(1342.345, -2) -- 1300.000
```



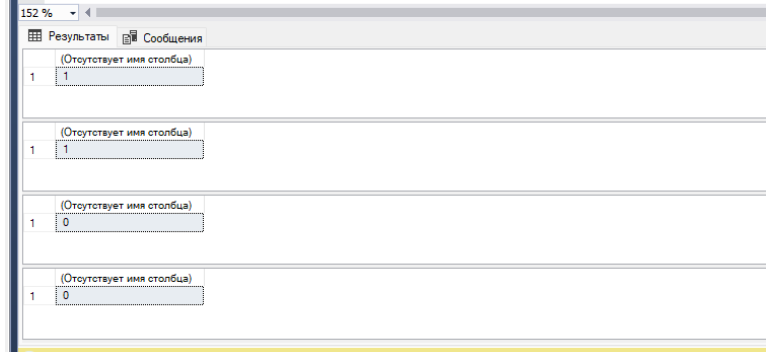
	(Отсутствует имя столбца)
1	1342.350

	(Отсутствует имя столбца)
1	1300.000

**ISNUMERIC:** определяет, является ли значение числом. В качестве параметра функция принимает выражение. Если выражение является числом, то функция возвращает 1. Если не является, то возвращается 0.

```
230 --ISNUMERIC
231 SELECT ISNUMERIC(1342.345) -- 1
232 SELECT ISNUMERIC('1342.345') -- 1
233 SELECT ISNUMERIC('SQL') -- 0
234 SELECT ISNUMERIC('13-04-2017') -- 0
```



	(Отсутствует имя столбца)
1	1

	(Отсутствует имя столбца)
1	1

	(Отсутствует имя столбца)
1	0

	(Отсутствует имя столбца)
1	0

```

236 --ABS: возвращает абсолютное значение числа.
237
238 SELECT ABS(-123)    -- 123
239
240 --CEILING: возвращает наименьшее целое число, которое больше или равно текущему значению.
241
242 SELECT CEILING(-123.45)  -- -123
243 SELECT CEILING(123.45)   -- 124
244
245 --LOOR: возвращает наибольшее целое число, которое меньше или равно текущему значению.
246 SELECT FLOOR(-123.45)   -- -124
247 SELECT FLOOR(123.45)    -- 123
248
249

```

152 %

Результаты	Сообщения
(Отсутствует имя столбца)	
1	123
(Отсутствует имя столбца)	
1	-123
(Отсутствует имя столбца)	
1	124
(Отсутствует имя столбца)	
1	-124
(Отсутствует имя столбца)	
1	123

```

249
250 --SQUARE: возводит число в квадрат.
251 SELECT SQUARE(5)    -- 25
252
253 --SQRT: получает квадратный корень числа.
254 SELECT SQRT(225)    -- 15
255
256 --RAND: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.
257 SELECT RAND()       -- 0.707365088352935
258 SELECT RAND()       -- 0.173808327956812

```

2 %

Результаты	Сообщения
(Отсутствует имя столбца)	
1	25
(Отсутствует имя столбца)	
1	15
(Отсутствует имя столбца)	
1	0.0879693157733806
(Отсутствует имя столбца)	
1	0.80609917695782

```

259
260 --COS: возвращает косинус угла, выраженного в радианах
261 SELECT COS(1.0472)  -- 0.5 - 60 градусов
262 --SIN: возвращает синус угла, выраженного в радианах
263 SELECT SIN(1.5708)  -- 1 - 90 градусов
264 --TAN: возвращает тангенс угла, выраженного в радианах
265 SELECT TAN(0.7854)  -- 1 - 45 градусов
266
267
268
269
270

```

152 %

Результаты	Сообщения
(Отсутствует имя столбца)	
1	0.499997879272546
(Отсутствует имя столбца)	
1	0.999999999993254
(Отсутствует имя столбца)	
1	1.00000367321185



Округлим произведение цены товара на количество этого товара:

```
266
267
268 --Округлим произведение цены товара на количество этого товара:
269 SELECT ProductName, ROUND(Price * ProductCount, 2)
270 FROM Products
271
```

152 %

Результаты   Сообщения

	ProductName	(Отсутствует имя столбца)
1	iPhone 6	72000,00
2	iPhone 6S	82000,00
3	iPhone 7	260000,00
4	Galaxy S8	92000,00
5	Galaxy S8 Plus	56000,00
6	Mi 5X	52000,00
7	OnePlus 5	228000,00

## T-SQL предоставляет ряд функций для работы с датами и временем:

Список часто используемых *функций времени и даты*:

GETDATE()	возвращает текущую дату и время
CURRENT_TIMEZONE()	возвращает имя часового пояса
GETUTCDATE()	возвращает текущую дату и время по Гринвичу (UTC/GMT)
DAY(дата)	возвращает день месяца указанной даты
MONTH(дата)	возвращает номер месяца указанной даты
YEAR(дата)	возвращает год указанной даты
DATEPART(часть, дата)	возвращает целое число, представляющее указанную часть заданной даты
DATENAME(часть, дата)	возвращает строку символов, представляющую указанную часть заданной даты
DATEADD(часть, число, дата)	добавляет указанное целое число со знаком к части входного значения даты, а затем возвращает это измененное значение
DATEDIFF(часть, начальная дата, конечная дата)	возвращает разницу как целое число со знаком между частями заданных дат
EOMONTH(дата)	возвращает последний день месяца, заданной даты

**GETDATE**: возвращает текущую локальную дату и время на основе системных часов в виде объекта datetime

```
SELECT GETDATE() -- 2017-07-28 21:34:55.830
```

**GETUTCDATE**: возвращает текущую локальную дату и время по гринвичу (UTC/GMT) в виде объекта datetime

```
SELECT GETUTCDATE() -- 2017-07-28 18:34:55.830
```

**SYSDATETIME** возвращает текущую локальную дату и время на основе системных часов, но отличие от **GETDATE** состоит в том, что дата и время возвращаются в виде объекта datetime2

```
SELECT SYSDATETIME() -- 2017-07-28 21:02:22.7446744
```

**SYSUTCDATETIME**: возвращает текущую локальную дату и время по гринвичу (UTC/GMT) в виде объекта datetime2

```
SELECT SYSUTCDATETIME() -- 2017-07-28 18:20:27.5202777
```

**SYSDATETIMEOFFSET**: возвращает объект datetimeoffset(7), который содержит дату и время относительно GMT

```
SELECT SYSDATETIMEOFFSET() -- 2017-07-28 21:02:22.7446744 +03:00
```

**DAY**: возвращает день даты, который передается в качестве параметра

```
SELECT DAY(GETDATE()) -- 28
```

**MONTH**: возвращает месяц даты

```
SELECT MONTH(GETDATE()) -- 7
```

**YEAR:** возвращает год из даты

***SELECT YEAR(GETDATE())*** -- 2017

**DATENAME:** возвращает часть даты в виде строки. Параметр выбора части даты передается в качестве первого параметра, а сама дата передается в качестве второго параметра:

***SELECT DATENAME(month, GETDATE())*** -- July

**Для определения части даты можно использовать следующие параметры (в скобках указаны их сокращенные версии):**

*year (yy, yyyy): год*  
*quarter (qq, q): квартал*  
*month (mm, m): месяц*  
*dayofyear (dy, y): день года*  
*day (dd, d): день месяца*  
*week (wk, ww): неделя*  
*weekday (dw): день недели*  
*hour (hh): час*  
*minute (mi, n): минута*  
*second (ss, s): секунда*  
*millisecond (ms): миллисекунда*  
*microsecond (mcs): микросекунда*  
*nanosecond (ns): наносекунда*  
*tzoffset (tz): смещение в минутах относительно гринвича (для объекта datetimeoffset)*

**DATEPART:** возвращает часть даты в виде числа. Параметр выбора части даты передается в качестве первого параметра (используются те же параметры, что и для DATENAME), а сама дата передается в качестве второго параметра:

**SELECT DATEPART(month, GETDATE())** -- 7

**DATEADD:** возвращает дату, которая является результатом сложения числа к определенному компоненту даты. Первый параметр представляет компонент даты, описанный выше для функции DATENAME. Второй параметр - добавляемое количество. Третий параметр - сама дата, к которой надо сделать прибавление:

**SELECT DATEADD(month, 2, '2017-7-28')** -- 2017-09-28 00:00:00.000

**SELECT DATEADD(day, 5, '2017-7-28')** -- 2017-08-02 00:00:00.000

**SELECT DATEADD(day, -5, '2017-7-28')** -- 2017-07-23 00:00:00.000

Если добавляемое количество представляет отрицательное число, то фактически происходит уменьшение даты.

**DATEDIFF:** возвращает разницу между двумя датами. Первый параметр - компонент даты, который указывает, в каких единицах стоит измерять разницу. Вторым и третьим параметрами - сравниваемые даты:

**SELECT DATEDIFF(year, '2017-7-28', '2018-9-28')** -- разница 1 год

**SELECT DATEDIFF(month, '2017-7-28', '2018-9-28')** -- разница 14 месяцев

**SELECT DATEDIFF(day, '2017-7-28', '2018-9-28')** -- разница 427 дней

**TODATETIMEOFFSET:** возвращает значение datetimeoffset, которое является результатом сложения временного смещения с объектом datetime2

**SELECT TODATETIMEOFFSET('2017-7-28 01:10:22', '+03:00')**



## **Преобразование данных**

Когда мы присваиваем значение одного типа столбцу, который хранит данные другого типа, либо выполняем операции, которые вовлекают данные разных типов, SQL Server пытается выполнить преобразование и привести используемое значение к нужному типу.

Но не все преобразования SQL Server может выполнить автоматически. SQL Server может выполнять неявные преобразования от типа с меньшим приоритетом к типу с большим приоритетом. Таблица приоритетов (чем выше, тем больший приоритет):

datetime  
smalldatetime  
float  
real  
decimal  
money  
smallmoney  
int  
smallint  
tinyint  
bit  
nvarchar  
nchar  
varchar  
char

То есть SQL Server автоматически может преобразовать число 100.0 (float) в дату и время (datetime).

В тех случаях, когда необходимо выполнить преобразования от типов с высшим приоритетом к типам с низшим приоритетом, то надо выполнять явное приведение типов. Для этого в T-SQL определены две функции: **CONVERT** и **CAST**.

Функция **CAST** преобразует выражение одного типа к другому. Она имеет следующую форму:

**CAST(выражение AS тип\_данных)**

Например, при выводе информации о заказах преобразует числовое значение и дату в строку:

```
1 SELECT Id, CAST(CreatedAt AS nvarchar) + '; total: ' + CAST(Price * ProductCount AS nvarchar)
2 FROM Orders
```

```
1 USE productsdb;
2
3 SELECT Id, CAST(CreatedAt AS nvarchar) + '; total: ' + CAST(Price * ProductCount AS nvarchar)
4 FROM Orders
```

100 %

Results Messages

	Id	(No column name)
1	1	2017-07-11; total: 96000.00
2	2	2017-07-13; total: 41000.00
3	3	2017-07-11; total: 41000.00





## Convert

Большую часть преобразований охватывает функция CAST. Если же необходимо какое-то дополнительное форматирование, то можно использовать функцию **CONVERT**. Она имеет следующую форму:

```
1 CONVERT(тип_данных, выражение [, стиль])
```

Третий необязательный параметр задает стиль форматирования данных. Этот параметр представляет числовое значение, которое для разных типов данных имеет разную интерпретацию. Например, некоторые значения для форматирования дат и времени:

- 0 или 100 - формат даты "Mon dd yyyy hh:miAM/PM" (значение по умолчанию)
- 1 или 101 - формат даты "mm/dd/yyyy"
- 3 или 103 - формат даты "dd/mm/yyyy"
- 7 или 107 - формат даты "Mon dd, yyyy hh:miAM/PM"
- 8 или 108 - формат даты "hh:mi:ss"
- 10 или 110 - формат даты "mm-dd-yyyy"
- 14 или 114 - формат даты "hh:mi:ss:mmm" (24-часовой формат времени)

Некоторые значения для форматирования данных типа money в строку:

- 0 - в дробной части числа остаются только две цифры (по умолчанию)
- 1 - в дробной части числа остаются только две цифры, а для разделения разрядов применяется запятая
- 2 - в дробной части числа остаются только четыре цифры

## Дополнительные функции

Кроме CAST, CONVERT, TRY\_CONVERT есть еще ряд функций, которые могут использоваться для преобразования в ряд типов:

- **STR(float [, length [, decimal]])**: преобразует число в строку. Второй параметр указывает на длину строки, а третий - сколько знаков в дробной части числа надо оставлять
- **CHAR(int)**: преобразует числовой код ASCII в символ. Нередко используется для тех ситуаций, когда необходим символ, который нельзя ввести с клавиатуры
- **ASCII(char)**: преобразует символ в числовой код ASCII
- **NCHAR(int)**: преобразует числовой код UNICODE в символ
- **UNICODE(char)**: преобразует символ в числовой код UNICODE

```
1 SELECT STR(123.4567, 6,2)  -- 123.46
2 SELECT CHAR(219)           -- Ы
3 SELECT ASCII('Ы')          -- 219
4 SELECT NCHAR(1067)         -- Ы
5 SELECT UNICODE('Ы')        -- 1067
```

## Функции CASE и IF

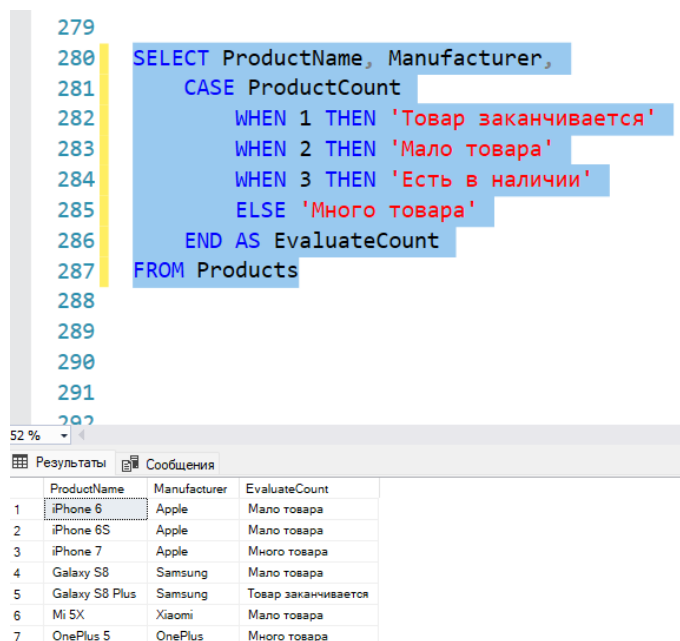
Функция CASE проверяет значение некоторого выражение, и в зависимости от результата проверки может возвращать тот или иной результат.

CASE принимает следующую форму:

*CASE выражение*

```
WHEN значение_1 THEN результат_1
WHEN значение_2 THEN результат_2
WHEN значение_N THEN результат_N
[ELSE альтернативный_результат]
```

*END*



```
279
280 SELECT ProductName, Manufacturer,
281        CASE ProductCount
282          WHEN 1 THEN 'Товар заканчивается'
283          WHEN 2 THEN 'Мало товара'
284          WHEN 3 THEN 'Есть в наличии'
285          ELSE 'Много товара'
286        END AS EvaluateCount
287 FROM Products
288
289
290
291
292
```

	ProductName	Manufacturer	EvaluateCount
1	iPhone 6	Apple	Мало товара
2	iPhone 6S	Apple	Мало товара
3	iPhone 7	Apple	Много товара
4	Galaxy S8	Samsung	Мало товара
5	Galaxy S8 Plus	Samsung	Товар заканчивается
6	Mi 5X	Xiaomi	Мало товара
7	OnePlus 5	OnePlus	Много товара

Здесь значения столбца *ProductCount* последовательно сравниваются со значениями после операторов *WHEN*. В зависимости от значения столбца *ProductCount* функция *CASE* будет возвращать одну из строк, которая идет после соответствующего оператора *THEN*. Для возвращаемого результата определен столбец *EvaluateCount*:

Также функция CASE может принимать еще одну форму:

### CASE

**WHEN выражение\_1 THEN результат\_1**  
**WHEN выражение\_2 THEN результат\_2**  
**WHEN выражение\_N THEN результат\_N**  
**[ELSE альтернативный\_результат]**  
**END**

```
288
289 SELECT ProductName, Manufacturer,
290        CASE
291            WHEN Price > 50000 THEN 'Категория А'
292            WHEN Price BETWEEN 40000 AND 50000 THEN 'Категория В'
293            WHEN Price BETWEEN 30000 AND 40000 THEN 'Категория С'
294            ELSE 'Категория D'
295        END AS Category
296 FROM Products
```

152 %

Результаты   Сообщения

	ProductName	Manufacturer	Category
1	iPhone 6	Apple	Категория С
2	iPhone 6S	Apple	Категория В
3	iPhone 7	Apple	Категория А
4	Galaxy S8	Samsung	Категория В
5	Galaxy S8 Plus	Samsung	Категория А
6	Mi 5X	Xiaomi	Категория D
7	OnePlus 5	OnePlus	Категория С

## ***IF***

***Функция IF в зависимости от результата условного выражения возвращает одно из двух значений. Общая форма функции выглядит следующим образом:***

***IF(условие, значение\_1, значение\_2)***

***Если условие в функции IF истинно то возвращается значение\_1, если ложно, то возвращается значение\_2.***

***Например:***

```
297
298 SELECT ProductName, Manufacturer,
299     IF(ProductCount>3, 'Много товара', 'Мало товара')
300 FROM Products
301
302
303
```

152 %

Результаты   Сообщения

	ProductName	Manufacturer	(Отсутствует имя столбца)
1	iPhone 6	Apple	Мало товара
2	iPhone 6S	Apple	Мало товара
3	iPhone 7	Apple	Много товара
4	Galaxy S8	Samsung	Мало товара
5	Galaxy S8 Plus	Samsung	Мало товара
6	Mi 5X	Xiaomi	Мало товара
7	OnePlus 5	OnePlus	Много товара

## Функции *NEWID*, *ISNULL* и *COALESCE*

### NEWID

Для генерации объекта UNIQUEIDENTIFIER, то есть некоторого уникального значения, используется функция **NEWID()**. Например, мы можем определить для столбца первичного ключа тип UNIQUEIDENTIFIER и по умолчанию присваивать ему значение функции NEWID:

```
1 CREATE TABLE Clients
2 (
3     Id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
4     FirstName NVARCHAR(20) NOT NULL,
5     LastName NVARCHAR(20) NOT NULL,
6     Phone NVARCHAR(20) NULL,
7     Email NVARCHAR(20) NULL
8 )
9
10 INSERT INTO Clients (FirstName, LastName, Phone, Email)
11 VALUES ('Tom', 'Smith', '+36436734', NULL),
12 ('Bob', 'Simpson', NULL, NULL)
```

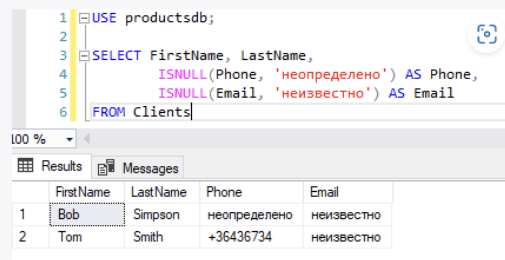
### ISNULL

Функция **ISNULL** проверяет значение некоторого выражения. Если оно равно NULL, то функция возвращает значение, которое передается в качестве второго параметра:

```
1 ISNULL(выражение, значение)
```

Например, возьмем выше созданную таблицу и применим при получении данных функцию ISNULL:

```
1 SELECT FirstName, LastName,
2     ISNULL(Phone, 'не определено') AS Phone,
3     ISNULL(Email, 'неизвестно') AS Email
4 FROM Clients
```



The screenshot shows a SQL query window with the following code:

```
1 USE productsdb;
2
3 SELECT FirstName, LastName,
4     ISNULL(Phone, 'неопределено') AS Phone,
5     ISNULL(Email, 'неизвестно') AS Email
6 FROM Clients
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	FirstName	LastName	Phone	Email
1	Bob	Simpson	неопределено	неизвестно
2	Tom	Smith	+36436734	неизвестно

## COALESCE

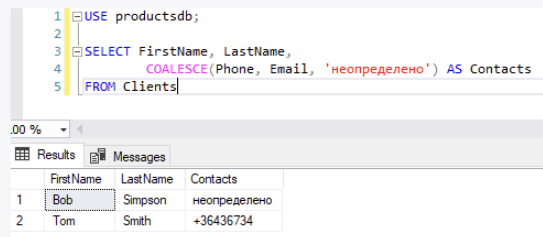
Функция **COALESCE** принимает список значений и возвращает первое из них, которое не равно NULL:

```
1 COALESCE(выражение_1, выражение_2, выражение_N)
```

Например, выберем из таблицы Clients пользователей и в контактах у них определим либо телефон, либо электронный адрес, если они не равны NULL:

```
1 SELECT FirstName, LastName,  
2     COALESCE(Phone, Email, 'не определено') AS Contacts  
3 FROM Clients
```

То есть в данном случае возвращается телефон, если он определен. Если он не определен, то возвращается электронный адрес. Если и электронный адрес не определен, то возвращается строка "не определено".



The screenshot shows a SQL query window with the following code:

```
1 USE productsdb;  
2  
3 SELECT FirstName, LastName,  
4     COALESCE(Phone, Email, 'неопределено') AS Contacts  
5 FROM Clients
```

Below the query window, the 'Results' tab is active, displaying the following data:

	FirstName	LastName	Contacts
1	Bob	Simpson	неопределено
2	Tom	Smith	+36436734