# **Встроенные функции**

## **Функции для работы со строками**

Для работы со строка в MySQL определен ряд встроенных функций:

* **CONCAT**: объединяет строки. В качестве параметра принимает от 2-х и более строк, которые надо соединить:

|  |  |
| --- | --- |
| 1 | SELECT CONCAT('Tom', ' ', 'Smith') -- Tom Smith |

* При этом в функцию можно передавать не только непосредственно строки, но и числа, даты - они будут преобразовываться в строки и также объединяться.
* **CONCAT\_WS**: также объединяет строки, но в качестве первого параметра принимает разделитель, который будет соединять строки:

|  |  |
| --- | --- |
| 1 | SELECT CONCAT\_WS(' ', 'Tom', 'Smith', 'Age:', 34) -- Tom Smith Age: 34 |

* **LENGTH**: возвращает количество символов в строке. В качестве параметра в функцию передается строка, для которой надо найти длину:

|  |  |
| --- | --- |
| 1 | SELECT LENGTH('Tom Smith') -- 9 |

* **LTRIM**: удаляет начальные пробелы из строки. В качестве параметра принимает строку:

|  |  |
| --- | --- |
| 1 | SELECT LTRIM(' Apple') |

* **RTRIM**: удаляет конечные пробелы из строки. В качестве параметра принимает строку:

|  |  |
| --- | --- |
| 1 | SELECT RTRIM(' Apple ') |

* **TRIM**: удаляет начальные и конечные пробелы из строки. В качестве параметра принимает строку:

|  |  |
| --- | --- |
| 1 | SELECT TRIM(' Tom Smith ') |

* С помощью дополнительного оператора можно задать где имеено удалить пробелы: BOTH (в начале и в конце), TRAILING (только в конце), LEADING (только в начале):

|  |  |
| --- | --- |
| 1 | SELECT TRIM(BOTH FROM ' Tom Smith ') |

* **LOCATE(find, search [, start])**: возвращает позицию первого вхождения подстроки find в строку search. Дополнительный параметр start позволяет установить позицию в строке search, с которой начинается поиск подстроки find. Если подстрока search не найдена, то возвращается 0:

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT LOCATE('om', 'Tom Smith'); -- 2  SELECT LOCATE('m', 'Tom Smith'); -- 3  SELECT LOCATE('m', 'Tom Smith', 4); -- 6  SELECT LOCATE('mig', 'Tom Smith'); -- 0 |

* **LEFT**: вырезает с начала строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

|  |  |
| --- | --- |
| 1 | SELECT LEFT('Apple', 3) -- App |

* **RIGHT**: вырезает с конца строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

|  |  |
| --- | --- |
| 1 | SELECT RIGHT('Apple', 3) -- ple |

* **SUBSTRING(str, start [, length])**: вырезает из строки str подстроку, начиная с позиции start. Третий необязательный параметр передает количество вырезаемых символов:

|  |  |
| --- | --- |
| 1  2 | SELECT SUBSTRING('Galaxy S8 Plus', 8), -- S8 Plus  (SELECT SUBSTRING('Galaxy S8 Plus', 8, 2) ); -- S8 |

* **SUBSTRING\_INDEX(str, delimiter, count)**: вырезает из строки str подстроку. Параметр delimiter определяет разделитель внутри строки. А параметр count определяет, до какого вхождения разделителя надо вырезать подстроку. Если count положительный, то подстрока вырезается с начала, если count отрицательный, то с конца строки str:

|  |  |
| --- | --- |
| 1  2  3 | SELECT SUBSTRING\_INDEX('Galaxy S8 Plus', ' ', 1), -- Galaxy  (SELECT SUBSTRING\_INDEX('Galaxy S8 Plus', ' ', 2) ), -- Galaxy S8  (SELECT SUBSTRING\_INDEX('Galaxy S8 Plus', ' ', -2) ); -- S8 Plus |

* **REPLACE(search, find, replace)**: заменяет в строке find подстроку search на подстроку replace. Первый параметр функции - строка, второй - подстрока, которую надо заменить, а третий - подстрока, на которую надо заменить:

|  |  |
| --- | --- |
| 1 | SELECT REPLACE('Galaxy S8 Plus', 'S8 Plus', 'Note 8') -- Galaxy Note 8 |

* **INSERT(str, start, length, insert)**: вставляет в строку str, заменяя length символов с позиции start подстрокой insert. Первый параметр функции - строка, второй - позиция, с которой надо заменить, третий - сколько символов с позиции start надо заменить вставляемой подстрокой, четвертый параметр - вставляемая подстрока:

|  |  |
| --- | --- |
| 1 | SELECT INSERT('Galaxy S9', 8, 3, 'Note 9'); -- Galaxy Note 9 |

* **REVERSE**: переворачивает строку наоборот:

|  |  |
| --- | --- |
| 1 | SELECT REVERSE('123456789') -- 987654321 |

* **LOWER**: переводит строку в нижний регистр:

|  |  |
| --- | --- |
| 1 | SELECT LOWER('Apple') -- apple |

* **UPPER**: переводит строку в верхний регистр

|  |  |
| --- | --- |
| 1 | SELECT UPPER('Apple') -- APPLE |

* **SPACE**: возвращает строку, которая содержит определенное количество пробелов
* **REPEAT(str, count)**: возвращает строку, которая содержит определенное количество повторов подстроки str. Количество повторов задается через параметр count.

|  |  |
| --- | --- |
| 1 | SELECT REPEAT('ab', 5); -- ababababab |

* **LPAD(str, length, pad)**: добавляет слева от строки str некоторое количество символов, которые определены в параметре pad. Количество добавляемых символов вычисляется по формуле length - LENGTH(str). Если параметр length меньше длины строки str, то эта строка усекается до length символов.

|  |  |
| --- | --- |
| 1 | SELECT LPAD('Tom Smith', 13, '\*'); -- \*\*\*\*Tom Smith |

* **RPAD(str, length, pad)**: добавляет справа от строки str некоторое количество символов, которые определены в параметре pad. Количество добавляемых символов вычисляется по формуле length - LENGTH(str). Если параметр length меньше длины строки str, то эта строка усекается до length символов.

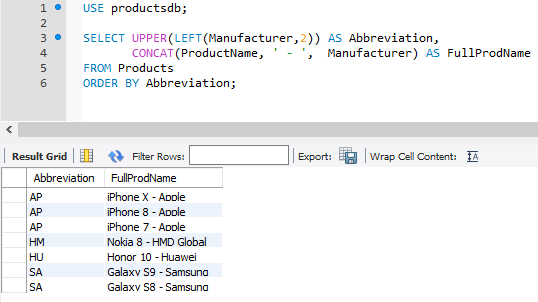
|  |  |
| --- | --- |
| 1 | SELECT RPAD('Tom Smith', 13, '\*'); -- Tom Smith\*\*\*\* |

Например, возьмем таблицу:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE Products  (  Id INT AUTO\_INCREMENT PRIMARY KEY,  ProductName VARCHAR(30) NOT NULL,  Manufacturer VARCHAR(20) NOT NULL,  ProductCount INT DEFAULT 0,  Price DECIMAL NOT NULL  ); |

И при извлечении данных применим строковые функции:

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT UPPER(LEFT(Manufacturer,2)) AS Abbreviation,  CONCAT(ProductName, ' - ', Manufacturer) AS FullProdName  FROM Products  ORDER BY Abbreviation |



## **Функции для работы с числами**

Для работы с числовыми данными MySQL предоставляет ряд функций:

* **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

|  |  |
| --- | --- |
| 1  2 | SELECT ROUND(1342.345, 2), -- 1342.35  (SELECT ROUND(1342.345, -2)); -- 1300; |

* **TRUNCATE**: оставляет в дробной части определенное количество символов

|  |  |
| --- | --- |
| 1 | SELECT TRUNCATE(1342.345, 2); -- 1342.34 |

* **ABS**: возвращает абсолютное значение числа.

|  |  |
| --- | --- |
| 1 | SELECT ABS(-123) -- 123 |

* **CEILING**: возвращает наименьшее целое число, которое больше или равно текущему значению.

|  |  |
| --- | --- |
| 1  2 | SELECT CEILING(-123.45), -- -123  (SELECT CEILING(123.45)); -- 124 |

* **FLOOR**: возвращает наибольшее целое число, которое меньше или равно текущему значению.

|  |  |
| --- | --- |
| 1  2 | SELECT FLOOR(-123.45), -- -124  (SELECT FLOOR(123.45)); -- 123 |

* **POWER**: возводит число в определенную степень.

|  |  |
| --- | --- |
| 1  2 | SELECT POWER(5, 2), -- 25  (SELECT POWER(5, 3)); -- 125 |

* **SQRT**: получает квадратный корень числа.

|  |  |
| --- | --- |
| 1 | SELECT SQRT(225); -- 15 |

* **SIGN**: возвращает -1, если число меньше 0, и возвращает 1, если число больше 0. Если число равно 0, то возвращает 0.

|  |  |
| --- | --- |
| 1  2 | SELECT SIGN(-5), -- -1  (SELECT SIGN(7)); -- 1 |

* **RAND**: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.

|  |  |
| --- | --- |
| 1  2 | SELECT RAND(); -- 0.707365088352935  SELECT RAND(); -- 0.173808327956812 |

Например, возьмем таблицу:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE Products  (  Id INT AUTO\_INCREMENT PRIMARY KEY,  ProductName VARCHAR(30) NOT NULL,  Manufacturer VARCHAR(20) NOT NULL,  ProductCount INT DEFAULT 0,  Price DECIMAL NOT NULL  ); |

Округлим произведение цены товара на количество этого товара:

|  |  |
| --- | --- |
| 1  2 | SELECT ProductName, ROUND(Price \* ProductCount, 2)  FROM Products; |

## **Функции для работы с датами и временем**

MySQL имеет ряд встроенных функций для работы с датами и временем.

#### **Получение даты и времени**

* Функции **NOW()**, **SYSDATE()**, **CURRENT\_TIMESTAMP()** возвращают текущую локальную дату и время на основе системных часов в виде объекта datetime. Все три функции возвращают одинаковый результат

|  |  |
| --- | --- |
| 1  2  3 | SELECT NOW(); -- 2018-05-25 21:34:55  SELECT SYSDATE(); -- 2018-05-25 21:34:55  SELECT CURRENT\_TIMESTAMP(); -- 2018-05-25 21:32:55 |

* Функции **CURDATE** и **CURRENT\_DATE** возвращают текущую локальную дату в виде объекта date:

|  |  |
| --- | --- |
| 1  2 | SELECT CURRENT\_DATE(); -- 2018-05-25  SELECT CURDATE(); -- 2018-05-25 |

* Функции **CURTIME** и **CURRENT\_TIME** возвращают текущее время в виде объекта time:

|  |  |
| --- | --- |
| 1  2 | SELECT CURRENT\_TIME(); -- 20:47:45  SELECT CURTIME(); -- 20:47:45 |

* **UTC\_DATE** возвращает текущую локальную дату относительно GMT

|  |  |
| --- | --- |
| 1 | SELECT UTC\_DATE(); -- 2018-05-25 |

* **UTC\_TIME** возвращает текущее локальное время относительно GMT

|  |  |
| --- | --- |
| 1 | SELECT UTC\_TIME(); -- 17:47:45 |

#### **Парсинг даты и времени**

* **DAYOFMONTH(date)** возвращает день месяца в виде числового значения
* **DAYOFWEEK(date)** возвращает день недели в виде числового значения
* **DAYOFYEAR(date)** возвращает номер дня в году
* **MONTH(date)** возвращает месяц даты
* **YEAR(date)** возвращает год из даты
* **QUARTER(date)** возвращает номер квартала года
* **WEEK(date [, first])** возвращает номер недели года. Необязательный параметр позволяет задать стартовый день недели. Если этот параметр равен 1, то первым днем считается понедельник, иначе воскресенье
* **LAST\_DAY(date)** возвращает последний день месяца в виде даты
* **DAYNAME(date)** возвращает название дня недели
* **MONTHNAME(date)** возвращает название текущего месяца
* **HOUR(time)** возвращает час времени
* **MINUTE(time)** возвращает минуту времени
* **SECOND(time)** возвращает секунду времени

Примеры функций:

|  |  |
| --- | --- |
| Функция | Результат |
| DAYOFMONTH('2018-05-25') | 25 |
| DAYOFWEEK('2018-05-25') | 6 |
| DAYOFYEAR('2018-05-25') | 145 |
| MONTH('2018-05-25') | 5 |
| YEAR('2018-05-25') | 2018 |
| QUARTER('2018-05-25') | 2 |
| WEEK('2018-05-25', 1) | 21 |
| LAST\_DAY('2018-05-25') | 2018-05-31 |
| DAYNAME('2018-05-25') | Friday |
| MONTHNAME('2018-05-25') | May |
| HOUR('21:25:54') | 21 |
| MINUTE('21:25:54') | 25 |
| SECOND('21:25:54') | 54 |

#### **Функция EXTRACT**

Функция EXTRACT извлекает из даты и времени какой-то определенный компонент. Ее формальный синтаксис:

|  |  |
| --- | --- |
| 1 | EXTRACT(unit FROM datetime) |

Значение datetime представляет исходную дату и (или) время, а значение unit указывает, какой компонент даты или времени будет извлекаться. Параметр unit может представлять одно из следующих значений:

* SECOND (секунды)
* MINUTE (минуты)
* HOUR (час)
* DAY (день)
* MONTH (месяц)
* YEAR (год)
* MINUTE\_SECOND (минуты и секунды)
* HOUR\_MINUTE (часы и минуты)
* DAY\_HOUR (день и часы)
* YEAR\_MONTH (год и месяц)
* HOUR\_SECOND (часы, минуты и секунды)
* DAY\_MINUTE (день, часы и минуты)
* DAY\_SECOND (день, чаы, минуты и секунды)

Примеры вызова функции:

|  |  |
| --- | --- |
| Вызов | Результат |
| EXTRACT( SECOND FROM '2018-05-25 21:25:54') | 54 |
| EXTRACT( MINUTE FROM '2018-05-25 21:25:54') | 25 |
| EXTRACT( HOUR FROM '2018-05-25 21:25:54') | 21 |
| EXTRACT( DAY FROM '2018-05-25 21:25:54') | 25 |
| EXTRACT( MONTH FROM '2018-05-25 21:25:54') | 5 |
| EXTRACT( YEAR FROM '2018-05-25 21:25:54') | 2018 |
| EXTRACT( MINUTE\_SECOND FROM '2018-05-25 21:25:54') | 2554 |
| EXTRACT( DAY\_HOUR FROM '2018-05-25 21:25:54') | 2521 |
| EXTRACT( YEAR\_MONTH FROM '2018-05-25 21:25:54') | 201805 |
| EXTRACT( HOUR\_SECOND FROM '2018-05-25 21:25:54') | 212554 |
| EXTRACT( DAY\_MINUTE FROM '2018-05-25 21:25:54') | 252125 |
| EXTRACT( DAY\_SECOND FROM '2018-05-25 21:25:54') | 25212554 |

#### **Функции для манипуляции с датами**

Ряд функций позволяют производить операции сложения и вычитания с датами и временем:

* **DATE\_ADD(date, INTERVAL expression unit)** возвращает объект DATE или DATETIME, который является результатом сложения даты date с определенным временным интервалом. Интервал задается с помощью выражения INTERVAL expression unit, где INTERVAL предоставляет ключевое слово, expression - количество добавляемых к дате единиц, а unit - тип единиц (часы, дни и т.д.) Параметр unit может иметь те же значения, что и в функции EXTRACT, то есть DAY, HOUR и т.д.
* **DATE\_SUB(date, INTERVAL expression unit)** возвращает объект DATE или DATETIME, который является результатом вычитания из даты date определенного временного интервала
* **DATEDIFF(date1, date2)** возвращает разницу в днях между датами date1 и date2
* **TO\_DAYS(date)** возвращает количество дней с 0-го года
* **TIME\_TO\_SEC(time)** возвращает количество секунд, прошедших с момента полуночи

Примеры применения:

|  |  |
| --- | --- |
| Вызов | Результат |
| DATE\_ADD('2018-05-25', INTERVAL 1 DAY) | 2018-05-26 |
| DATE\_ADD('2018-05-25', INTERVAL 3 MONTH) | 2018-08-25 |
| DATE\_ADD('2018-05-25 21:31:27', INTERVAL 4 HOUR) | 2018-05-26 01:31:27 |
| DATE\_SUB('2018-05-25', INTERVAL 4 DAY) | 2018-05-21 |
| DATEDIFF('2018-05-25', '2018-05-27') | -2 |
| DATEDIFF('2018-05-25', '2018-05-21') | 4 |
| DATEDIFF('2018-05-25', '2018-03-21') | 65 |
| TO\_DAYS('2018-05-25') | 737204 |
| TIME\_TO\_SEC('10:00') | 36000 |

#### **Форматирование дат и времени**

* **DATE\_FORMAT(date, format)** возвращает объект DATE или DATETIME, отформатированный с помощью шаблона format
* **TIME\_FORMAT(date, format)** возвращает объект TIME или DATETIME, отформатированный с помощью шаблона format

Обе функции в качестве второго параметра принимают строку форматирования или шаблон, который показывает, как оформатировать значение. Этот шаблон может принимать следующие значения:

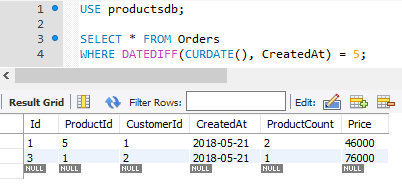
* %m: месяц в числовом формате 01..12
* %с: месяц в числовом формате 1..12
* %M: название месяца (January...December)
* %b: аббревиатура месяца (Jan...Dec)
* %d: день месяца в числовом формате 00..31
* %e: день месяца в числовом формате 0..31
* %D: номер дня месяца с суффиксом (1st, 2nd, 3rd...)
* %y: год в виде двух чисел
* %Y: год в виде четырех чисел
* %W: название дня недели (Sunday...Saturday)
* %a: аббревиатура дня недели (Sun...Sat)
* %H: час в формате 00..23
* %k: час в формате 0..23
* %h: час в формате 01..12
* %l: час в формате 1..12
* %i: минуты в формате 00..59
* %r: время в 12-ти часовом формате (hh:mm:ss AM или PM)
* %T: время в 24-ти часовом формате (hh:mm:ss)
* %S: секунды в формате 00..59
* %p: AM или PM

Примеры применения:

|  |  |
| --- | --- |
| Вызов | Результат |
| DATE\_FORMAT('2018-05-25', '%d/%m/%y') | 25/05/18 |
| DATE\_FORMAT('2018-05-25 21:25:54', '%d %M %Y') | 25 May 2018 |
| DATE\_FORMAT('2018-05-25 21:25:54', '%r') | 09:25:54 PM |
| TIME\_FORMAT('2018-05-25 21:25:54', '%H:%i:%S') | 21:25:24 |
| TIME\_FORMAT('21:25:54', '%k:%i') | 21:25 |

В качестве примера использования функций найдем заказы, которые были сделаны 5 дней назад:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Orders  WHERE DATEDIFF(CURDATE(), CreatedAt) = 5; |



## **Функции CASE, IF, IFNULL, COALESCE**

### **CASE**

Функция **CASE** проверяет истинность набора условий и в зависимости от результата проверки может возвращать тот или иной результат. Эта функция принимает следующую форму:

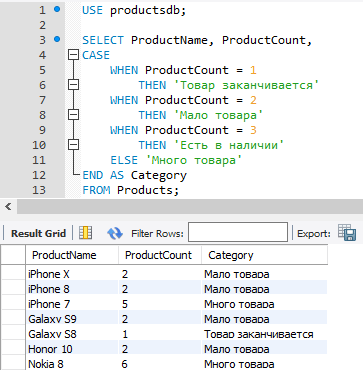
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CASE  WHEN условие\_1 THEN результат\_1  WHEN условие\_2 THEN результат\_2  .................................  WHEN условие\_N THEN условие\_N  [ELSE альтернативный\_результат]  END |

Возьмем для примера следующую таблицу Products:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE Products  (  Id INT AUTO\_INCREMENT PRIMARY KEY,  ProductName VARCHAR(30) NOT NULL,  Manufacturer VARCHAR(20) NOT NULL,  ProductCount INT DEFAULT 0,  Price DECIMAL NOT NULL  ); |

Выполним запрос к этой таблице и используем функцию CASE:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | SELECT ProductName, ProductCount,  CASE  WHEN ProductCount = 1  THEN 'Товар заканчивается'  WHEN ProductCount = 2  THEN 'Мало товара'  WHEN ProductCount = 3  THEN 'Есть в наличии'  ELSE 'Много товара'  END AS Category  FROM Products; |



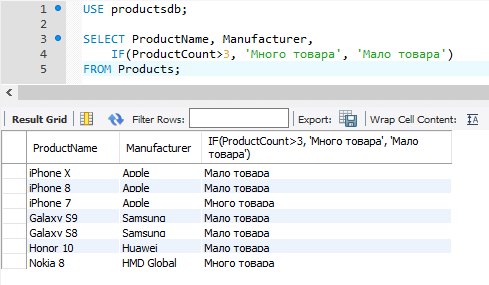
### **Функция IF**

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

|  |  |
| --- | --- |
| 1 | IF(условие, значение\_1, значение\_2) |

Если условие, передаваемое в качестве первого параметра, верно, то возвращается первое значение, иначе возвращается второе значение. Например:

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, Manufacturer,  IF(ProductCount > 3, 'Много товара', 'Мало товара')  FROM Products; |



### **IFNULL**

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

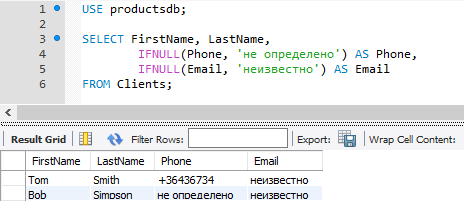
|  |  |
| --- | --- |
| 1 | IFNULL(выражение, значение) |

Например, возьмем следующую таблицу

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | CREATE TABLE Clients  (  Id INT AUTO\_INCREMENT PRIMARY KEY,  FirstName VARCHAR(20) NOT NULL,  LastName VARCHAR(20) NOT NULL,  Phone VARCHAR(20) NULL,  Email VARCHAR(20) NULL  );    INSERT INTO Clients (FirstName, LastName, Phone, Email)  VALUES ('Tom', 'Smith', '+36436734', NULL),  ('Bob', 'Simpson', NULL, NULL); |

И применим при получении данных функцию IFNULL:

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT FirstName, LastName,  IFNULL(Phone, 'не определено') AS Phone,  IFNULL(Email, 'неизвестно') AS Email  FROM Clients; |



### **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  2  3 | SELECT FirstName, LastName,  COALESCE(Phone, Email, 'не определено') AS Contacts  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".

# Функции

Создание функции:

delimiter //

CREATE FUNCTION имя\_функции(параметры)

RETURNS возращаемый\_тип

BEGIN

Тело\_функции;

RETURN возвращаемое\_значение;

END //

delimiter ;

Вызвать функцию можно в выражении SELECT:

SELECT имя\_функции(фактические\_параметры).

Для того чтобы удалить фукнцию, нужно выполнить:

DROP FUNCTION имя\_функции.

Объявление и инициализация переменных, циклы, условия не отличаются от тех, что используются в хранимых процедурах.

*Пример:*

delimiter //

CREATE FUNCTION GetSum (val1 INT, val2 INT)

RETURNS INT

BEGIN

DECLARE sum INT;

SET sum = val1 + val2;

RETURN sum;

END //

delimiter ;

SELECT GetSum(2, 3);

# Регулярные выражения

Регулярное выражение – это шаблон, который обработчик регулярных выражений пытается сопоставить с введенным текстом. Шаблон состоит из односимвольных или многосимвольных литералов, операторов или конструкций.

**Зачем нужны регулярные выражения?**

* **Сравнение с шаблоном**: Регулярные выражения отлично помогают определять, соответствует ли строка тому или иному формату – например, телефонному номеру, адресу электронной почты или номеру кредитной карты.
* **Замена**: При помощи регулярных выражений легко находить и заменять шаблоны в строке.
* **Извлечение**: При помощи регулярных выражений легко извлекать из шаблона фрагменты информации.
* **Портируемость**: Почти в любом распространенном языке программирования есть своя библиотека регулярных выражений. Синтаксис в основном стандартизирован, поэтому вам не придется переучиваться регулярным выражениям при переходе на новый язык.

**С чего начать?**

Преимущество использования регулярного выражения заключается в том, что вы не ограничены поиском строки на основе фиксированного шаблона со знаком процента (%) и подчеркиванием (\_) в операторе *LIKE*. Регулярные выражения имеют больше метасимволов для создания гибких шаблонов.

Недостаток использования регулярных выражений заключается в том, что довольно сложно понять и поддерживать такой сложный шаблон. Поэтому вы должны описать значение регулярного выражения в комментарии к оператору *SQL*. Кроме того, скорость извлечения данных в некоторых случаях уменьшается, если вы используете сложные шаблоны в регулярном выражении.

Простейшее регулярное выражение не включает в себя специальных символов. Например, регулярное выражение *hello* означает совпадение с *hello* и ничего больше.

Следующее иллюстрирует синтаксис оператора *REGEXP* в предложении *WHERE*:

*SELECT Названия\_столбцов*

*FROM Название\_таблицы*

*WHERE Название\_столбца REGEXP шаблон;*

Если значение совпадает с шаблоном, выражение в предложении *WHERE* возвращает *true*, в противном случае возвращает *false*.

Форма отрицания оператора *REGEXP* есть *NOT* *REGEXP*.

Пример:

*SELECT* ‘*hello*’ *REGEXP* ‘*hello*’; --> *true*

**Метасимволы в регулярных выражениях:**

|  |  |  |
| --- | --- | --- |
| Метасимвол | Описание | Пример |
| . | Соответствует одному любому символу. | а.б.в соответствует строкам а1б2в или айбив. |
| [] | Содержат отдельные символы и диапазоны значений, в числе которых должно встречаться проверяемое значение. | [абв] соответствует строке, содержащей а, б или в. |
| ^ | Если стоит в начале выражения, то следующий символ должен соответствовать началу строки. Если заключено в квадратные скобки, то проверяемое значение не должно соответствовать последующим в скобках символам.  Примечание:  Если предшествует квадратным скобкам, то начало строки проверяется на соответствие этим скобкам. | [^абв] соответствует строкам, не содержащим сочетания букв а, б и в. Проверку пройдут строки гав и бах, но не бав. ^[абв] соответствует строке, начинающейся с а, б или в. |
| - | Указывает диапазон, в котором должно содержаться проверяемое значение.  Примечание:  Диапазон должен быть заключен в квадратные скобки. | [1–5] соответствует строкам 12345 или 26589, но не 6789. |
| ? | Предшествующие символы или диапазоны значений | Окт? соответствует Окт или Октябрь, но не Декабрь. |
| + | Предшествующие символы или диапазоны значений могут иметь соответствие в строке один или несколько раз. | [0–9]+ соответствует 1, 11, 456 и т.д. |
| \* | Предшествующие символы или диапазоны значений могут иметь соответствие в строке один или несколько раз, или не иметь ни одного соответствия. | 12\*3 соответствует 1223, 13 и 123, но не 223 или 23. |
| () | Содержит группу выражений и значений.  Можно группировать выражения или последовательности. | (кат) соответствует словам категория и конкатенация, но не карта. |
| $ | Предыдущий символ должен соответствовать концу строки. | [123]$ соответствует строке, содержащей в конце 1, 2 или 3. |
| | | Означает альтернативную фразу или написание. | М|международный соответствует Международный и международный. |

# Хранимые процедуры

Хранимые процедуры представляет набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект. Изменится процесс покупки товара, соответственно достаточно будет изменить код процедуры. То есть процедура также упрощает управление кодом.

Хранимые процедуры обычно выполняются быстрее, чем обычные *SQL*-инструкции. Все потому что код процедур компилируется один раз при первом ее запуске, а затем сохраняется в скомпилированной форме.

Процедуры могут принимать параметры. Параметры бывают входными (*in*) – с их помощью в процедуру можно передать некоторые значения. И также параметры бывают выходными (*out*) – они позволяют возвратить из процедуры некоторое значение. По умолчанию параметры являются входными.

**Как создать процедуру?**

*delimiter* //

*CREATE PROCEDURE* имя\_процедуры(параметры)  
*BEGIN*

Тело\_процедуры;  
*END* //

*delimiter* ;

Для вызова процедуры используется команда *CALL* имя\_процедуры(фактический\_параметр1, фактический\_параметр2, … )

Чтобы получить значение выходного параметра, можно указать в качестве выходного параметра при вызове процедуры @имя\_переменной, и затем выбрать её с помощью выражения *SELECT*.

**Объявление переменной:**

*DECLARE* имя\_переменной тип\_переменной;

**Присваивание значения:**

*SET* имя\_переменной = значение\_переменной;

В качестве значения переменной можно использовать и подзапрос.

**Условие:**

*IF* условие *THEN*

действие;

*ELSE*

действие;

***END IF*;**

**Цикл:**

*WHILE* условие *DO*

действие;

*END* *WHILE*;

Для того, чтобы удалить процедуру, нужно выполнить: *DROP PROCEDURE* имя\_процедуры.

Пример:

*delimiter* //

*CREATE PROCEDURE* *OutputString*(*IN* *str VARCHAR*(20))  
*BEGIN*

*SELECT str*;  
*END* //

*delimiter* ;

*CALL OutputString*(‘*string’*);