

Mysql-Func&Operator

[零 参考资料 2](#_Toc66112764)

[一 运算符 2](#_Toc66112765)

[1 概述 2](#_Toc66112766)

[2 比较运算符 3](#_Toc66112767)

[二 Control Flow Functions 5](#_Toc66112768)

[三 数字相关 Functions & Operator 6](#_Toc66112769)

[1 计算运算符 6](#_Toc66112770)

[2 Function 7](#_Toc66112771)

[四 Date and Time function 10](#_Toc66112772)

[1 概述 10](#_Toc66112773)

[2 时区相关： 10](#_Toc66112774)

[3 具体函数 10](#_Toc66112775)

[4 日期时间 FORMAT 占位符 18](#_Toc66112776)

[5 时间单位 18](#_Toc66112777)

[6 GET\_FORMAT 19](#_Toc66112778)

[7 星期第一天 19](#_Toc66112779)

[五 String Function and operators 19](#_Toc66112780)

[1 概述 19](#_Toc66112781)

[2 函数 19](#_Toc66112782)

[3 地区列表 27](#_Toc66112783)

[六 Aggregate Functions 29](#_Toc66112784)

[七 Cast Functions and Operators 31](#_Toc66112785)

[八 XML Function 32](#_Toc66112786)

[九 Bit Functions and Operators 32](#_Toc66112787)

[一〇 Encryption and Compression Functions 32](#_Toc66112788)

[一一 Locking Functions 32](#_Toc66112789)

[一二 Information Functions 33](#_Toc66112790)

[一三 Miscellaneous FUnctions 33](#_Toc66112791)

# 零 参考资料

《高性能Mysql 第三版》

官方文档：

<https://dev.mysql.com/doc/refman/5.7/en/functions.html>

<https://dev.mysql.com/doc/refman/5.7/en/sql-statements.html>

# 运算符

<https://dev.mysql.com/doc/refman/5.7/en/non-typed-operators.html>

## 概述

* + - 1. 操作符 =，用于赋值（assignment）还是比较(comparison),优先级不同
      2. 如果同级别优先级，赋值时从右往左，其他情况从左往右

1. --优先级从上到下依次递减
2. INTERVAL
3. **BINARY**, **COLLATE**
4. !
5. - (unary minus), ~ (unary **bit** inversion)
6. ^
7. \*, /, DIV, %, MOD
8. -, +
9. <<, >>
10. &
11. |
12. = (comparison), <=>, >=, >, <=, <, <>, !=, **IS**, LIKE, REGEXP, IN
13. BETWEEN, CASE, **WHEN**, **THEN**, **ELSE**
14. NOT
15. AND, &&
16. XOR
17. OR, ||
18. = (assignment), :=

|  |  |
| --- | --- |
| 运算符 | 描述 |
| 位运算 | |
| & | 位运算，与 |
| >> | 右移位 |
| << | 左移位 |
| %, MOD | 取模 |
| \* | 乘法 |
| + | 加法 |
| - | 减法 |
| - |  |
| -> | 获取指定JSON column值的指定属性; equivalent to JSON\_EXTRACT(). |
| ->> | 获取指定JSON column值的指定属性; equivalent to JSON\_UNQUOTE(JSON\_EXTRACT()).在5.7.13引入 |
| / | 除法 |
| 逻辑条件运算符 | |
| AND, && | 逻辑，与 |
| OR | 逻辑运算，或 |
| NOT, ! | 否定运算符  默认情况下，！的优先级高于NOT  HIGH\_NOT\_PRECEDENCE启用时，两则优先级相同 |
| || | 默认情况下，位逻辑运算符或，同 OR运算符；  当PIPES\_AS\_CONCAT启用时，则为字符串连接符 |
| XOR | 逻辑运算，非或 |
| 其他 | |
| := | 赋值 |
| = | 赋值，用于SET语句 |
| = | 等于 |
| ^ | 位运算，异或 |
| BINARY | 将字符串转成二进制字符串 |
| CASE | Case |
| DIV | 整数除法 |
| REGEXP | 正则 |
| NOT REGEXP | 不匹配正则 |
| RLIKE | Whether string matches regular expression |
| SOUNDS LIKE | Compare sounds |
| | | 位运算，或 |
| ~ | 位运算，取反 |

## 比较运算符

* + - 1. 比较类运算符，结果有1(TRUE)，0(FALSE)，NULL，除了LEAST() 和 GREATEST()
      2. 会根据实际值，数值和字符串会自动转换
      3. 比较符 =,>,<,>=,<=,<>,!= 不仅能用于值比较，还可以用于row comparisons
      4. 默认情况下，字符串比较大小写不敏感

|  |  |
| --- | --- |
| 运算符 | 描述 |
| > | 1. 大于 2. 支持row comparisons即：   (a, b) > (x, y)  等价于  (a > x) OR ((a = x) AND (b > y)) |
| >= | 1. 大于等于    1. 支持row comparisons即：   (a, b) >= (x, y)  等价于  (a > x) OR ((a = x) AND (b >= y)) |
| < | 小于  支持row comparisons即：  (a, b) < (x, y)  等价于  (a < x) OR ((a = x) AND (b < y)) |
| <>, != | 不等于  支持row comparisons即：  (a, b) <> (x, y) 或者(a, b) != (x, y)  等价于  (a <> x) OR (b <> y) |
| <= | 小于等于  支持row comparisons即：  (a, b) <= (x, y)  等价于  (a < x) OR ((a = x) AND (b <= y)) |
| <=> | NULL-safe equal.  具体效果如下   1. **SELECT** 1 <=> 1, NULL <=> NULL, 1 <=> NULL; 2. -> 1, 1, 0 3. **SELECT** 1 = 1, NULL = NULL, 1 = NULL; 4. -> 1, NULL, NULL   支持row comparisons即：  (a, b) <=> (x, y)  等价于  (a <=> x) AND (b <=> y) |
| = | 等于 |
| ***expr*** BETWEEN ***A*** AND ***B*** | 大于等于A，小于等于B之间的值, |
| ***expr*** NOT BETWEEN ***A*** AND ***B*** | 不在大于等于A，小于等于B之间的值 |
| ***expr*** IN() | 1. 值是否在指定集合内，实际判断，通过二进制值判断 2. IN的元素个数由max\_allowed\_packet参数限制，默认4MB 3. 比较结果返回NULL情况   a ***expr***为 NULL  b IN中元素没有匹配值，且其中一个元素为NULL   1. 支持row comparisons即 2. **SELECT** (3,4) IN ((1,2), (3,4)); 3. -> 1 4. **SELECT** (3,4) IN ((1,2), (3,5)); 5. -> 0   5.隐式转换的坑：   1. --这里字符会被转为浮点型0.0,所以结果都为1(TRUE) 2. **SELECT** 'a' IN (0), 0 IN ('b'); 3. -> 1, 1 |
| ***expr*** NOT IN() | 不在指定集合，其他同IN |
| ***A*** IS ***B*** | 判断值是否符合，B的结果值可以是1(TRUE)，0(FALSE)，UNKNOWN   1. **SELECT** 1 **IS** **TRUE**, 0 **IS** **FALSE**, NULL **IS** UNKNOWN; 2. -> 1, 1, 1 |
| IS NULL | 1判断值是否NULL  2对于自增字段：  2.1 如果sql\_auto\_is\_null参数为1，则下面语句会返回最后插入的记录  **SELECT** \* **FROM** tbl\_name **WHERE** auto\_col **IS** NULL  2.2 如果sql\_auto\_is\_null参数为0，则不返回数据  3 对于DATE和DATETIME的非空字段，对应的0值数据，可以通过下面方式查找：  SELECT \* FROM tbl\_name WHERE date\_column IS NULL |
| IS NOT | 判断值是否不符合，其他与IS 相同 |
| IS NOT NULL | 判断值是否非NULL，其他与IS NULL相同 |
| LIKE | 简单的模式匹配 |
| NOT LIKE | 不符合简单的模式匹配 |
| COALESCE(...) | 返回集合中第一个非NULL值，如果集合中都是NULL，则返回NULL |
| GREATEST(...) | 返回集合中最大的值；  如果是字符串，则按照指定的字符集排序规则获取最大值；  如果集合中都是NULL，则返回NULL |
| INTERVAL(N，N1，N2...) | 通过二进制比较，在N1，N2，N3。。。集合中找到第一个大于N的值下标  下标从0开始，即如果N1>N，则返回0，如果N2>N返回1  如果N为NULL或找不到，返回-1 |
| ISNULL(expr) | 如果expr结果值异常或为NULL，则返回1，其他情况返回0   1. **SELECT** ISNULL(1+1); 2. -> 0 3. **SELECT** ISNULL(1/0); 4. -> 1   因为值与NULL比较，结果总为NULL，所以，可以用ISNULL代替  其他大部分行为跟IS NULL 相同 |
| LEAST(v1,v2,v3...) | 获取集合中最小的值  如果集合元素都是NULL，返回NULL  如果元素都是整数，按整数比较  如果元素有一个是双精度浮点型，都转为转双精度浮点型比较  如果元素有一个是DECIMAL，都转为转DECIMAL比较  如果元素是数值和字符混合，则都按数值比较  如果元素都是非二进制字符串，则都按非二进制字符串比较  如果其他情况，都按对应的二进制字符串比较 |

# Control Flow Functions

|  |  |
| --- | --- |
| **Name** | **Description** |
| CASE | CASE [value] WHEN [condition | compareValue] THEN result [WHEN [condition | compareValue] THEN result ...] [ELSE result] END  1 当指定value时，返回符合第一个WHEN CompareValue 对应的result，如果都不符合WHEN，则如果有指定ELSE result 则返回ELSE指定的result，否则返回NULL  2 当没有指定value时，返回符合第一个WHEN condition对应的result，如果都不符合WHEN，则如果有指定ELSE result 则返回ELSE指定的result，否则返回NULL |
| IF() | IF(expr1,expr2,expr3) 等同于 expr1?expr2:expr3  依次遵循以下规则  1 只有expr2或expr3显示指定NULL时，才有可能返回NULL  2 如果expr2和expr3有一个是String类型，返回值都是String  3 如果expr2和expr3都是String类型其中一个是大小写敏感的，则返回值都是大小写敏感的string  4 如果expr2和expr3有一个是floating-point类型，返回值都是floating-point  5 如果expr2和expr3有一个是整数，返回值都是整数 |
| IFNULL() | IFNULL(expr1,expr2) 等同于expr1 !=NULL ？expr1: expr2  实际返回类型，按兼容expr1和expr2的通用类型返回，依次为string，real，integer |
| NULLIF() | NULLIF(expr1,expr2) 等同于expr1==expr2？NULL:expr1  返回值类型同expr1 |

# 数字相关 Functions & Operator

<https://dev.mysql.com/doc/refman/5.7/en/numeric-functions.html>

## 计算运算符

* + - 1. 计算结果遵循以下规则
         1. 如果两个数字都是integer，对于 - + \* 运算符，结果均为BIGINT(64 bit)类型
         2. 如果两个数字都是integer且其中一个是unsigned，则结果为unsigned integer
         3. 如果SQL mode 是NO\_UNSIGNED\_SUBTRACTION，则对于减法，无论两个整数数字是否unsigned，其结果都是signed
         4. 对于 +, -, /, \*, % 运算符，两个数字其中一个是real或string类型，则结果的精度为能支持结果的最大精度
         5. 对于 / 运算符，结果的小数点长度为，第一个计算数的小数点位数 加上 系统参数div\_precision\_increment的值(默认为4)，例子：5.05 / 0.014 = 360.714286
         6. a.5的规则适用于计算过程中每一步，例子(14620/9432456)/(24250/9432456)=(0.0014)/(0.0026)= 0.60288653

|  |  |
| --- | --- |
| 运算符 | 描述 |
| %, MOD | 取模，等同 MOD() 方法 |
| \* | 乘法 |
| + | 加法 |
| - | 减法 |
| - | 负号，尽量避免对BIGINT使用 |
| / | 除法  1 除0时，返回NULL  2 仅当需要强转为integer时，才用BIGINT类型进行计算 |
| DIV | 整数除法  1 如果两个数至少有一个非integer，则将两个数字转为Decimal类型进行计算，并将结果转成BIGINT  2 如果结果值超过了BIGINT范围，将报错 |

## Function

|  |  |
| --- | --- |
| Name | **Description** |
| ABS() | ABS(X)  返回绝对值，返回值类型为BIGINT |
| ACOS() | ACOS(X)  1 反余弦函数  2 如果x非-1 to 1，返回NULL |
| ASIN() | ASIN(X)  1反正弦函数  2 如果x非-1 to 1，返回NULL |
| ATAN() | ATAN (X) |
| ATAN2()  ATAN() | ATAN(Y,X), ATAN2(Y,X) |
| CEIL() | CEIL(X),同CEILING() |
| CEILING() | CEILING(X)  1 返回大于X的最小整数  2 如果X为字符串或floating-point类型，则返回值类型为floating-point |
| CONV() | CONV(N,from\_base,to\_base)  1 将N值从from\_base转到to\_base进制  2 基于64bit计算  3 如果任一个参数为NULL，返回NULL  4 from\_base/to\_base的最小值为2，最大值为36   1. SELECT CONV('a',16,2); 2. -> '1010' 3. SELECT CONV('6E',18,8); 4. -> '172' 5. SELECT CONV(-17,10,-18); 6. -> '-H' 7. SELECT CONV(10+'10'+'10'+X'0a',10,10); 8. -> '40' |
| COS() | COS(X)  余弦函数 |
| COT() | COT(X)  余切函数 |
| CRC32() | CRC32(expr)  1 获取expr的CRC32码  2 如果expr为NULL，则返回NULL  3 expr值必须是String，如果不是，则会被转为String进行处理 |
| DEGREES() | DEGREES(X)  将弧度X转为角度   1. SELECT DEGREES(PI()); 2. -> 180 |
| EXP() | EXP(X)  返回e的X次方值   1. SELECT EXP(2); 2. -> 7.3890560989307 |
| FLOOR() | FLOOR(X)  返回小于X的最大整数  1 如果X是字符串或floating-point类型，返回值为floating-point类型  2 其他情况返回值类型同X类型 |
| LN() | LN(X)  1返回X的自然对数  2 如果X小于等于0，则返回NULL，并抛出WARNING |
| LOG() | LOG(X)  同EXP(X)  LOG(B,X)  1 等同于log**B**X  2 如果B小于等于1，返回NULL   1. **SELECT** LOG(2,65536); 2. -> 16 |
| LOG10() | LOG10(X)  log**10**X  如果X小于等于0，返回NULL并抛出WARNING |
| LOG2() | LOG2(X)  log**2**X  如果X小于等于0，返回NULL并抛出WARNING |
| MOD() | MOD(N,M)  等同N%M  M为0时，返回NULL |
| PI() | 返回π值，默认返回7位小数   1. **SELECT** PI(); 2. -> 3.141593 3. **SELECT** PI()+0.000000000000000000; 4. -> 3.141592653589793116 |
| POW() | POW(X,Y)  即X**Y** |
| POWER() | 同POW() |
| RADIANS() | RADIANS(X)  将角度X转为弧度 |
| RAND() | RAND([N])  1 如果未指定N，则随机返回一个floating-point v值，0<=v<1  2 如果指定N，N将被作为种子值 |
| ROUND() | 1如果X值为精准数字，则结果为四舍五入(“round half away from zero” or “round toward nearest”)  2如果X值为近似值，则依赖当前系统采用的C语言，取最近的偶数(round to nearest even)   1. **SELECT** ROUND(2.5), ROUND(25E-1); 2. +------------+--------------+ 3. | ROUND(2.5) | ROUND(25E-1) | 4. +------------+--------------+ 5. | 3          |            2 | 6. +------------+--------------+   **ROUND(X)**  1等同于ROUND(X,0)  2返回值类型跟X相同  **ROUND(X,D)**   1. 返回值类型跟X相同 2. D如果没有指定，默认为0 3. D如果指定则保留指定位数的小数部分 4. D最大值为30，最小值为-30，如果超出范围，则超出部分被截取 5. **SELECT** ROUND(-1.23); 6. -> -1 7. **SELECT** ROUND(-1.58); 8. -> -2 9. **SELECT** ROUND(1.58); 10. -> 2 11. **SELECT** ROUND(1.298, 1); 12. -> 1.3 13. **SELECT** ROUND(1.298, 0); 14. -> 1 15. **SELECT** ROUND(23.298, -1); 16. -> 20 |
| SIGN() | SIGN(X)  1 判断X为正负数  2 X=负数返回 -1，X=0返回0，X=正数返回1 |
| SIN() | SIN(X)  返回弧度为X的sin值 |
| SQRT() | SQRT(X)  1 返回非负数的X的平方根  2 如果X为负数，则返回NULL |
| TAN() | TAN(X)  返回弧度为X的tan值 |
| TRUNCATE() | TRUNCATE(X,D)  1 如果D>0 ，保留小数点后面D位小数  2 如果D=0，则去掉小数点后面数值  3 如果D<0,则从小数点开始，从右往左用0替代D位数值   1. **SELECT** **TRUNCATE**(1.223,1); 2. -> 1.2 3. **SELECT** **TRUNCATE**(1.999,1); 4. -> 1.9 5. **SELECT** **TRUNCATE**(1.999,0); 6. -> 1 7. **SELECT** **TRUNCATE**(-1.999,1); 8. -> -1.9 9. **SELECT** **TRUNCATE**(122,-2); 10. -> 100 11. **SELECT** **TRUNCATE**(10.28\*100,0); 12. -> 1028 |

# Date and Time function

<https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html>

## 概述

* + - 1. 如NOW()、CURDATE()、CURTIME()、UTC\_DATE()、UTC\_TIME()、UTC\_TIMESTAMP()以及它们的同义词，每次query请求，每相同的Function只执行一次，且以第一次执行结果为准，即如果有多个NOW()，则只以第一个NOW()计算结果为准
      2. 如CURRENT\_TIMESTAMP()、CURRENT\_TIME()、CURRENT\_DATE()、FROM\_UNIXTIME()返回值依赖当前session所处时区。

## 时区相关：

<https://dev.mysql.com/doc/refman/5.7/en/time-zone-support.html>

## 具体函数

|  |  |
| --- | --- |
| Name | **Description** |
| ADDDATE() | 同义词：DATE\_ADD()  ADDDATE(date,INTERVAL expr UNIT)  1、将date加上指定单位量  2、UNIT取值：  <https://dev.mysql.com/doc/refman/5.7/en/expressions.html#temporal-intervals>   1. **SELECT** ADDDATE('2008-01-02', INTERVAL 31 DAY); 2. -> '2008-02-02'   ADDDATE(expr,days)  1、将expr 加上days天数  2、等同ADDDATE('2008-01-02', INTERVAL 31 DAY)   1. **SELECT** ADDDATE('2008-01-02', 31); 2. -> '2008-02-02' |
| ADDTIME() | ADDTIME(expr1,expr2)  将时间或日期(expr1),加上时间(expr2)   1. **SELECT** ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002'); 2. -> '2008-01-02 01:01:01.000001' 3. **SELECT** ADDTIME('01:00:00.999999', '02:00:00.999998'); 4. -> '03:00:01.999997' |
| CONVERT\_TZ() | CONVERT\_TZ(dt,from\_tz,to\_tz)  1 将日期dt，从from\_tz 时区，转成to\_tz时区的时间  2 如果参数不合法，将返回NULL  3 如果to\_tz为UTC，且结果值超过了UTC范围，则不发生转换操作 |
| CURDATE() | CURDATE()  返回YYYY-MM-DD 或YYYYMMDD格式的当前日期   1. **SELECT** CURDATE(); 2. -> '2008-06-13' 3. **SELECT** CURDATE() + 0; 4. -> 20080613 |
| CURRENT\_DATE()  CURRENT\_DATE | 同义词：CURDATE() |
| CURRENT\_TIME()  CURRENT\_TIME | 同义词：CURTIME()  CURRENT\_TIME, CURRENT\_TIME([fsp]) |
| CURRENT\_TIMESTAMP()  CURRENT\_TIMESTAMP | 同义词 NOW()  CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP([fsp]) |
| CURTIME() | CURTIME([fsp])  1 返回'hh:mm:ss' 或 hhmmss的当前时间  2 通过fsp指定时间小数点位数，默认0，取值范围：0-6   1. **SELECT** CURTIME(); 2. -> '23:50:26' 3. **SELECT** CURTIME() + 0; 4. -> 235026.000000 |
| DATE() | DATE(expr)  获取指定日期时间expr中的日期   1. **SELECT** **DATE**('2003-12-31 01:02:03'); 2. -> '2003-12-31' |
| DATE\_ADD() | DATE\_ADD(date,INTERVAL expr unit)  1 将指定日期/日期时间，加上指定单位unit的expr量  2 unit取值  <https://dev.mysql.com/doc/refman/5.7/en/expressions.html#temporal-intervals>  3 expr实际计算时按string处理  4 如果expr为负数，则以相反计算，比如加 -> 减，减 -> 加  5 如果date是  6 unit 取值参考5.6 |
| DATE\_FORMAT() | DATE\_FORMAT(date,format)  将date日期，格式化为format样式  返回字符串，可以包含非ASCII字符  format样式占位符参考 5.5   1. **SELECT** DATE\_FORMAT('2009-10-04 22:23:00', '%W %M %Y'); 2. -> 'Sunday October 2009' 3. **SELECT** DATE\_FORMAT('2007-10-04 22:23:00', '%H:%i:%s'); 4. -> '22:23:00' 5. **SELECT** DATE\_FORMAT('1900-10-04 22:23:00', '%D %y %a %d %m %b %j'); 6. -> '4th 00 Thu 04 10 Oct 277' 7. **SELECT** DATE\_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w'); 8. -> '22 22 10 10:23:00 PM 22:23:00 00 6' 9. **SELECT** DATE\_FORMAT('1999-01-01', '%X %V'); 10. -> '1998 52' 11. **SELECT** DATE\_FORMAT('2006-06-00', '%d'); 12. -> '00' |
| DATE\_SUB() | DATE\_SUB(date,INTERVAL expr unit)  1将指定日期/日期时间，减去指定单位unit的expr量   1. 其他特性同DATE\_ADD 2. unit 取值参考5.6 |
| DATEDIFF() | DATEDIFF(expr1,expr2)  1计算日期时间expr1 – 日期时间expr2的天数  2 即使expr1或expr2包含时间，只计算日期部分 |
| DAY() | 同DAYOFMONTH() |
| DAYNAME() | DAYNAME(date)  返回指定日期的星期几的本地语言名，具体根据系统参数lc\_time\_names确定   1. **SELECT** DAYNAME('2007-02-03'); 2. -> 'Saturday' 3. **SET** lc\_time\_names = 'es\_MX'; 4. **SELECT** DAYNAME('2010-01-01'); 5. -> viernes |
| DAYOFMONTH() | DAYOFMONTH(date)  返回指定日期的日，数字表示(0-31) |
| DAYOFWEEK() | DAYOFWEEK(date)  返回指定日期是星期几(1 = Sunday, 2 = Monday, …, 7 = Saturday)  符合ODBC标准 |
| DAYOFYEAR() | DAYOFYEAR(date)  返回指定日期是当年第几天(1-366) |
| EXTRACT() | EXTRACT(unit FROM date)  从date中提取指定数据(unit)  unit 取值参考5.6   1. mysql> **SELECT** EXTRACT(YEAR **FROM** '2019-07-02'); 2. -> 2019 3. mysql> **SELECT** EXTRACT(YEAR\_MONTH **FROM** '2019-07-02 01:02:03'); 4. -> 201907 5. mysql> **SELECT** EXTRACT(DAY\_MINUTE **FROM** '2019-07-02 01:02:03'); 6. -> 20102 7. mysql> **SELECT** EXTRACT(MICROSECOND **FROM** '2003-01-02 10:30:00.000123'); 8. -> 123 |
| FROM\_DAYS() | FROM\_DAYS(N)   1. **SELECT** FROM\_DAYS(730669); 2. -> '2000-07-03' |
| FROM\_UNIXTIME() | FROM\_UNIXTIME(unix\_timestamp[,format])  将时间戳unix\_timestamp按照format格式展示  format 默认为YYYY-MM-DD hh:mm:ss 格式，占位符参考5.5 |
| GET\_FORMAT() | GET\_FORMAT({DATE|TIME|DATETIME},{'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})  获取指定默认日期格式，具体参考5.7   1. **SELECT** DATE\_FORMAT('2003-10-03',GET\_FORMAT(**DATE**,'EUR')); 2. -> '03.10.2003' 3. **SELECT** STR\_TO\_DATE('10.31.2003',GET\_FORMAT(**DATE**,'USA')); 4. -> '2003-10-31' |
| HOUR() | HOUR(time)  返回指定时间的小时  小时值，实际可以大于23   1. **SELECT** **HOUR**('10:05:03'); 2. -> 10 3. **SELECT** **HOUR**('272:59:59'); 4. -> 272 |
| LAST\_DAY | LAST\_DAY(date)  返回指定日期的当月最后一天   1. **SELECT** LAST\_DAY('2003-02-05'); 2. -> '2003-02-28' 3. **SELECT** LAST\_DAY('2004-02-05'); 4. -> '2004-02-29' 5. **SELECT** LAST\_DAY('2004-01-01 01:01:01'); 6. -> '2004-01-31' 7. **SELECT** LAST\_DAY('2003-03-32'); 8. -> NULL |
| LOCALTIME()  LOCALTIME | Synonym for NOW() |
| LOCALTIMESTAMP  LOCALTIMESTAMP() | Synonym for NOW() |
| MAKEDATE() | MAKEDATE(year,dayofyear)  获取指定年份year的第dayofyear天的日期   1. **SELECT** MAKEDATE(2011,31), MAKEDATE(2011,32); 2. -> '2011-01-31', '2011-02-01' 3. **SELECT** MAKEDATE(2011,365), MAKEDATE(2014,365); 4. -> '2011-12-31', '2014-12-31' 5. **SELECT** MAKEDATE(2011,0); 6. -> NULL |
| MAKETIME() | MAKETIME(hour,minute,second)  生成时分秒时间，second可以有小数部分  SELECT MAKETIME(12,15,30);  -> '12:15:30' |
| MICROSECOND() | MICROSECOND(expr)  返回微秒部分   1. **SELECT** MICROSECOND('12:00:00.123456'); 2. -> 123456 |
| MINUTE() | MINUTE(time)  返回分钟 |
| MONTH() | MONTH(date)  返回月 |
| MONTHNAME() | MONTHNAME(date)  返回月英文名   1. **SELECT** MONTHNAME('2008-02-03'); 2. -> 'February' |
| NOW() | NOW([fsp])  返回当前时间  1 默认是返回YYYY-MM-DD hh:mm:ss  2 如果+0，则返回YYYYMMDDhhmmss  3 如果指定fsp，则在尾部加上微秒   1. 注意，如果一个语句中有多个NOW(),则所有的NOW()值，以第一个为准 2. **SELECT** NOW(); 3. -> '2007-12-15 23:50:26' 4. **SELECT** NOW() + 0; 5. -> 20071215235026.000000 6. **SELECT** NOW(), SLEEP(2), NOW(); 7. +---------------------+----------+---------------------+ 8. | NOW()               | SLEEP(2) | NOW()               | 9. +---------------------+----------+---------------------+ 10. | 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 | 11. +---------------------+----------+---------------------+ |
| PERIOD\_ADD() | PERIOD\_ADD(P,N)  对日期P（格式为YYMM或YYYYMM）加N个月，返回格式为YYYYMM |
| PERIOD\_DIFF() | PERIOD\_DIFF(P1,P2)  返回相差月份，P1&P2格式必须是YYMM或YYYYMM   1. **SELECT** PERIOD\_DIFF(200802,200703); 2. -> 11 |
| QUARTER() | QUARTER(date)  返回日期处在自然年中的第几季节，返回值1 to 4   1. **SELECT** QUARTER('2008-04-01'); 2. -> 2 |
| SEC\_TO\_TIME() | SEC\_TO\_TIME(seconds)  将秒seconds转为时间，返回默认格式是HH:mm:ss  如果 +0，则返回HHmmss格式的整数   1. **SELECT** SEC\_TO\_TIME(2378); 2. -> '00:39:38' 3. **SELECT** SEC\_TO\_TIME(2378) + 0; 4. -> 3938 |
| SECOND() | SECOND(time)  返回秒 |
| STR\_TO\_DATE() | STR\_TO\_DATE(str,format)  将字符串str按format 格式转成日期  1 如果结果包含日期和时间，则返回DATETIME类型  2 如果结果仅包含日期，则返回DATE类型  3 如果结果仅包含时间，则返回TIME类型  4 如果str非法，则返回NULL并抛出WARN  5 str和format格式前匹配，并忽略额外前置字符   1. **SELECT** STR\_TO\_DATE('a09:30:17','a%h:%i:%s'); 2. -> '09:30:17' 3. **SELECT** STR\_TO\_DATE('a09:30:17','%h:%i:%s'); 4. -> NULL 5. mysql> **SELECT** STR\_TO\_DATE('09:30:17a','%h:%i:%s'); 6. -> '09:30:17'   6 str和format格式匹配，但未指定所有占位符，则未指定的占位符则为0   1. **SELECT** STR\_TO\_DATE('abc','abc'); 2. -> '0000-00-00' 3. **SELECT** STR\_TO\_DATE('9','%m'); 4. -> '0000-09-00' 5. **SELECT** STR\_TO\_DATE('9','%s'); 6. -> '00:00:09'   7 如果sql\_mode = 'NO\_ZERO\_DATE'，则0值返回NULL   1. **SELECT** STR\_TO\_DATE('00/00/0000', '%m/%d/%Y'); 2. -> NULL |
| SUBDATE() | SUBDATE(date,INTERVAL expr unit)  SUBDATE(date,expr)  等同于DATE\_SUB()  将日期date，减去指定时间长度expr  expr默认为天数，可通过指定时间单位unit  unit 取值参考5.6 |
| SUBTIME() | SUBTIME(expr1,expr2)  计算日期时间expr1- 日期时间expr2之间的差值  返回值类型根据expr1值类型决定   1. **SELECT** SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002'); 2. -> '2007-12-30 22:58:58.999997' 3. **SELECT** SUBTIME('01:00:00.999999', '02:00:00.999998'); 4. -> '-00:59:59.999999' |
| SYSDATE() | SYSDATE([fsp])  返回当前日期时间，通过fsp指定微秒长度  与NOW()不同，如果sql中有多个SYSDATE，返回值为各自调用时的当前时间 |
| TIME() | TIME(expr)  获取指定日期时间expr中的时间  SELECT TIME('2003-12-31 01:02:03');  -> '01:02:03'  SELECT TIME('2003-12-31 01:02:03.000123');  -> '01:02:03.000123' |
| TIME\_FORMAT() | TIME\_FORMAT(time,format)  将时间time按format格式化  format取值参考5.4 |
| TIME\_TO\_SEC() | TIME\_TO\_SEC(time)  获取时间time的秒数 |
| TIMEDIFF() | TIMEDIFF(expr1,expr2)  计算日期时间expr1-日期时间expr2 的时间差异，返回类型是TIME  expr1和expr2格式必须一样   1. **SELECT** TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001'); 2. -> '-00:00:00.000001' 3. **SELECT** TIMEDIFF('2008-12-31 23:59:59.000001', '2008-12-30 01:01:01.000002'); 4. -> '46:58:57.999999' |
| TIMESTAMP() | TIMESTAMP(expr1),  TIMESTAMP(expr1,expr2)  默认情况下返回expr的日期时间格式  如果指定了expr2，则返回expr1+expr2的结果日期时间   1. **SELECT** **TIMESTAMP**('2003-12-31'); 2. -> '2003-12-31 00:00:00' 3. **SELECT** **TIMESTAMP**('2003-12-31 12:00:00','12:00:00'); 4. -> '2004-01-01 00:00:00' |
| TIMESTAMPADD() | TIMESTAMPADD(unit,interval,datetime\_expr)  将日期时间datetime\_expr，加上指定interval个unit时间单位  unit 取值参考5.6 |
| TIMESTAMPDIFF() | TIMESTAMPDIFF(unit,datetime\_expr1,datetime\_expr2)  计算datetime\_expr2 − datetime\_expr1的unit时间单位差   1. **SELECT** TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01'); 2. -> 3 3. **SELECT** TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01'); 4. -> -1 5. **SELECT** TIMESTAMPDIFF(**MINUTE**,'2003-02-01','2003-05-01 12:05:55'); 6. -> 128885 |
| TO\_DAYS() | TO\_DAYS(date)  获取从1582年开始到日期date之间的天数  SELECT TO\_DAYS(950501);  -> 728779  SELECT TO\_DAYS('2007-10-07');  -> 73332  SELECT TO\_DAYS('2008-10-07'), TO\_DAYS('08-10-07');  -> 733687, 733687  SELECT TO\_DAYS('0000-00-00');  -> null |
| TO\_SECONDS() | TO\_SECONDS(expr)  获取从1582年开始到日期date之间的秒数  特性同TO\_DAYS() |
| UNIX\_TIMESTAMP() | UNIX\_TIMESTAMP([date])  返回1970-01-01 00:00:00至date的秒数  如果date不指定，则默认为到当前时间的秒数   1. **SELECT** UNIX\_TIMESTAMP(); 2. -> 1447431666 3. **SELECT** UNIX\_TIMESTAMP('2015-11-13 10:20:19'); 4. -> 1447431619 5. **SELECT** UNIX\_TIMESTAMP('2015-11-13 10:20:19.012'); 6. -> 1447431619.012 |
| UTC\_DATE() | UTC\_DATE  UTC\_DATE()  获取当前UTC日期   1. **SELECT** UTC\_DATE(), UTC\_DATE() + 0; 2. -> '2003-08-14', 20030814 |
| UTC\_TIME() | UTC\_TIME  UTC\_TIME([fsp])  获取当前UTC时间   1. **SELECT** UTC\_TIME(), UTC\_TIME() + 0; 2. -> '18:07:53', 180753.000000 |
| UTC\_TIMESTAMP() | UTC\_TIMESTAMP  UTC\_TIMESTAMP([fsp])  返回当前UTC的日期时间，可以通过fsp指定微秒位数 |
| WEEK() | WEEK(date[,mode])  返回日期date在当前年的第几周  mode取值参考5.7 |
| WEEKDAY() | WEEKDAY(date)  返回日期date是星期几，0 = Monday, 1 = Tuesday, … 6 = Sunday |
| WEEKOFYEAR() | WEEKOFYEAR(date)  返回日期date是当前年的第几周，  等同于WEEK(date,3) |
| YEAR() | YEAR(date)  获取日期date的年 |
| YEARWEEK() | YEARWEEK(date),  YEARWEEK(date,mode)  返回日期date对应的年、第几周  未指定mode，则等同YEARWEEK(date,0)   1. **SELECT** YEARWEEK('1987-01-01'); 2. -> 198652 |

## 日期时间 FORMAT 占位符

|  |  |
| --- | --- |
| 占位符 | 描述 |
| %a | 星期英文缩写(Sun..Sat) |
| %b | 月份 英文缩写 (Jan..Dec) |
| %c | 月份，数字表示 (0..12) |
| %D | 日，英文后缀表示 (0th, 1st, 2nd, 3rd, …) |
| %d | 日, 双位数字表示 (00..31) |
| %e | 日, 数字表示 (0..31) |
| %f | 微秒 (000000..999999) |
| %H | 小时（24小时），双数字表示(00..23) |
| %h | 小时（12小时），双数字表示 (01..12) |
| %I | 同%H |
| %i | 分, 双数字表示 (00..59) |
| %j | 年中第几天，三数字表示 (001..366) |
| %k | 小时（24小时），数字表示(0..23) |
| %l | 小时（12小时），数字表示(1..12) |
| %M | 月，英文名 (January..December) |
| %m | 月, 双数字表示 (00..12) |
| %p | AM or PM |
| %r | 当前时间, 12-hour (*hh:mm:ss* followed by AM or PM) |
| %S | 秒，双数字表示 (00..59) |
| %s | 同%S |
| %T | 当前时间, 24-hour (*hh:mm:ss*) |
| %U | 获取当前是今年第几周(00..53)，星期日是一周的第一天; [WEEK()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_week) mode 0 |
| %u | 获取当前是今年第几周(00..53)，星期一是一周的第一天； [WEEK()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_week) mode 1 |
| %V | 获取当前是今年第几周(01..53), 星期日是一周的第一天; [WEEK()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_week) mode 2; 跟 %X 配合 |
| %v | 获取当前是今年第几周(01..53), 星期一是一周的第一天; [WEEK()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_week) mode 3; 跟 %x 配合使用 |
| %W | 星期几英文名 (Sunday..Saturday) |
| %w | 星期几，数字表示 (0=Sunday..6=Saturday) |
| %X | 年，星期日是一周的第一天，四位数; 跟%V 配合使用  SELECT DATE\_FORMAT("2019-01-01", "%X%V") AS currentWeek;  -> 201852 |
| %x | 年，星期一是一周的第一天，四位数，跟%v 配合使用 |
| %Y | 自然年, 四位数字 |
| %y | 自然年, 两位数字 |
| %% | % 字符 |
| %*x* | *x*, for any “*x*” not listed above |

## 时间单位

|  |  |
| --- | --- |
| ***unit* Value** | **Expected *expr* Format** |
| MICROSECOND | MICROSECONDS |
| SECOND | SECONDS |
| MINUTE | MINUTES |
| HOUR | HOURS |
| DAY | DAYS |
| WEEK | WEEKS |
| MONTH | MONTHS |
| QUARTER | QUARTERS |
| YEAR | YEARS |
| SECOND\_MICROSECOND | 'SECONDS.MICROSECONDS' |
| MINUTE\_MICROSECOND | 'MINUTES:SECONDS.MICROSECONDS' |
| MINUTE\_SECOND | 'MINUTES:SECONDS' |
| HOUR\_MICROSECOND | 'HOURS:MINUTES:SECONDS.MICROSECONDS' |
| HOUR\_SECOND | 'HOURS:MINUTES:SECONDS' |
| HOUR\_MINUTE | 'HOURS:MINUTES' |
| DAY\_MICROSECOND | 'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS' |
| DAY\_SECOND | 'DAYS HOURS:MINUTES:SECONDS' |
| DAY\_MINUTE | 'DAYS HOURS:MINUTES' |
| DAY\_HOUR | 'DAYS HOURS' |
| YEAR\_MONTH | 'YEARS-MONTHS' |

## GET\_FORMAT

|  |  |
| --- | --- |
| **Function Call** | **Result** |
| [GET\_FORMAT(DATE,'USA')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%m.%d.%Y' |
| [GET\_FORMAT(DATE,'JIS')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y-%m-%d' |
| [GET\_FORMAT(DATE,'ISO')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y-%m-%d' |
| [GET\_FORMAT(DATE,'EUR')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%d.%m.%Y' |
| [GET\_FORMAT(DATE,'INTERNAL')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y%m%d' |
| [GET\_FORMAT(DATETIME,'USA')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y-%m-%d %H.%i.%s' |
| [GET\_FORMAT(DATETIME,'JIS')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y-%m-%d %H:%i:%s' |
| [GET\_FORMAT(DATETIME,'ISO')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y-%m-%d %H:%i:%s' |
| [GET\_FORMAT(DATETIME,'EUR')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y-%m-%d %H.%i.%s' |
| [GET\_FORMAT(DATETIME,'INTERNAL')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%Y%m%d%H%i%s' |
| [GET\_FORMAT(TIME,'USA')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%h:%i:%s %p' |
| [GET\_FORMAT(TIME,'JIS')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%H:%i:%s' |
| [GET\_FORMAT(TIME,'ISO')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%H:%i:%s' |
| [GET\_FORMAT(TIME,'EUR')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%H.%i.%s' |
| [GET\_FORMAT(TIME,'INTERNAL')](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html#function_get-format) | '%H%i%s' |

## 星期第一天

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **First day of week** | **Range** | **Week 1 is the first week …** |
| **0** | Sunday | 0-53 | with a Sunday in this year |
| **1** | Monday | 0-53 | with 4 or more days this year |
| **2** | Sunday | 1-53 | with a Sunday in this year |
| **3** | Monday | 1-53 | with 4 or more days this year |
| **4** | Sunday | 0-53 | with 4 or more days this year |
| **5** | Monday | 0-53 | with a Monday in this year |
| **6** | Sunday | 1-53 | with 4 or more days this year |
| **7** | Monday | 1-53 | with a Monday in this year |

# String Function and operators

## 概述

## 函数

|  |  |
| --- | --- |
| Name | **Description** |
| LIKE | expr LIKE [charset\_name] pat [ESCAPE 'escape\_char'] [COLLATE charset\_name]  判断expar是否匹配排除escape\_char字符的pat  可通过charset\_name，指定字符集  除非数据类型为binary或大小写敏感的字符集，否则大小写不敏感   |  |  |  | | --- | --- | --- | | 字符 | 意义 | 转义 | | \_ | 一个任意字符占位符 | \\_ | | % | 任意个任意字符占位符 | \% |  1. **SELECT** 'David\_' LIKE 'David|\_' **ESCAPE** '|'; 2. -> 1 3. **SELECT** 'David!' LIKE 'David\_'; 4. -> 1 5. **SELECT** 'David!' LIKE '%D%v%'; 6. -> 1 7. **SELECT** 'David!' LIKE 'David\\_'; 8. -> 0 9. **SELECT** 'David\_' LIKE 'David\\_'; 10. -> 1 11. **SELECT** 'abc' LIKE 'ABC'; 12. -> 1 13. **SELECT** 'abc' LIKE \_latin1 'ABC' **COLLATE** latin1\_general\_cs; 14. -> 0 15. **SELECT** 'abc' LIKE \_latin1 'ABC' **COLLATE** latin1\_bin; 16. -> 0 17. **SELECT** 'abc' LIKE **BINARY** 'ABC'; 18. -> 0 19. **SELECT** 10 LIKE '1%'; 20. -> 1 |
| NOT LIKE | expr LIKE [charset\_name] pat [ESCAPE 'escape\_char'] [COLLATE charset\_name]  性质同LIKE |
| STRCMP() | STRCMP(expr1 [COLLATE charset\_name],expr2 [COLLATE charset\_name])  比较expr1,expr2两个字符串  如果相同则返回0  如果expr1在字符排序规则中排序靠后，则返回-1，否则返回1   1. **SELECT** STRCMP('text', 'text2'); 2. -> -1 3. **SELECT** STRCMP('text2', 'text'); 4. -> 1 5. **SELECT** STRCMP('text', 'text'); 6. -> 0 8. **SET** @s1 = \_latin1 'x' **COLLATE** latin1\_general\_ci; 9. **SET** @s2 = \_latin1 'X' **COLLATE** latin1\_general\_ci;  //大写X 10. **SET** @s3 = \_latin1 'x' **COLLATE** latin1\_general\_cs; 11. **SET** @s4 = \_latin1 'X' **COLLATE** latin1\_general\_cs;  //大写X 12. **SELECT** STRCMP(@s1, @s2), STRCMP(@s3, @s4); 13. -> 0，1 14. **SELECT** STRCMP(@s1, @s3 **COLLATE** latin1\_general\_ci); 15. -> 0 |
| ASCII() | ASCII(str)  获取str最右边字符的ascii值  str最长8 bit char  如果str=null，返回null  如果str='',返回0 |
| BIN() | BIN(N)  获取N的二进制表达  N最大值为BIGINT.MAX  如果N=null，返回null |
| BIT\_LENGTH() | BIT\_LENGTH(str)  返回str的位长度   1. **SELECT** BIT\_LENGTH('text'); 2. -> 32 |
| CHAR() | CHAR(N,... [USING charset\_name])  将ASCII码数组N转成对应的字符串  通过USING指定使用的字符集charset\_name  默认是binary  N元素最大值255   1. **SELECT** **CHAR**(77,121,83,81,'76'); 2. -> 'MySQL' 3. **SELECT** **CHAR**(77,77.3,'77.3'); 4. -> 'MMM' 5. **SELECT** HEX(**CHAR**(1,0)), HEX(**CHAR**(256)); 6. ->0100，0100 |
| CHAR\_LENGTH() | CHAR\_LENGTH(str)  获取str的字符长度 |
| CHARACTER\_LENGTH() | 等同CHAR\_LENGTH() |
| CONCAT() | CONCAT(str1,str2,...)  拼接字符串  如果其中一个参数为NULL，则返回NULL |
| CONCAT\_WS() | CONCAT\_WS(separator,str1,str2,...)  将多个字符串通过分隔符separator拼接  如果一个参数为NULL，则忽略，但不会忽略空字符串   1. **SELECT** CONCAT\_WS(',','First name','Second name','Last Name'); 2. -> 'First name,Second name,Last Name' 3. **SELECT** CONCAT\_WS(',','First name',NULL,'Last Name'); 4. -> 'First name,Last Name' |
| ELT() | ELT(N,str1,str2,str3,...)  返回第N个str，N 取值从1开始  如果N<1或越界，则返回NULL   1. **SELECT** ELT(1, 'Aa', 'Bb', 'Cc', 'Dd'); 2. -> 'Aa' 3. **SELECT** ELT(4, 'Aa', 'Bb', 'Cc', 'Dd'); 4. -> 'Dd' |
| EXPORT\_SET() | EXPORT\_SET(bits,on,off[,separator[,number\_of\_bits]])  返回bits的二进制表示的每bit值，如果当前bit为1，则设为on值，否则设置为off值，通过separator拼接  如果实际长度不足number\_of\_bits，则用off值填充  number\_of\_bits如果没指定，默认为最大值64，如果超过64，则等同64   1. **SELECT** EXPORT\_SET(5,'Y','N',',',4); 2. -> 'Y,N,Y,N' 3. **SELECT** EXPORT\_SET(6,'1','0',',',10); 4. -> '0,1,1,0,0,0,0,0,0,0' |
| FIELD() | FIELD(str,str1,str2,str3,...)  返回str在list中的第一个位置，下标从1开始，如果没有，则返回0  如果str=null，则返回0   1. **SELECT** FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff'); 2. -> 2 3. **SELECT** FIELD('Gg', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff'); 4. -> 0 |
| FIND\_IN\_SET() | FIND\_IN\_SET(str,strlist)  返回str在strlist中的第一个位置，下标从1开始，如果没有，则返回0  如果str=null，则返回null   1. **SELECT** FIND\_IN\_SET('b','a,b,c,d'); 2. -> 2 |
| FORMAT() | FORMAT(X,D[,locale])  将X四舍五入保留D位小数，不足部分用0填充  locale取值参考6.3   1. **SELECT** FORMAT(12332.123456, 4); 2. -> '12,332.1235' 3. **SELECT** FORMAT(12332.1,4); 4. -> '12,332.1000' 5. **SELECT** FORMAT(12332.2,0); 6. -> '12,332' 7. **SELECT** FORMAT(12332.2,2,'de\_DE'); 8. -> '12.332,20' |
| FROM\_BASE64() | FROM\_BASE64(str)  解码base64字符串str  如果str=null，返回null   1. **SELECT** TO\_BASE64('abc'), FROM\_BASE64(TO\_BASE64('abc')); 2. -> 'JWJj', 'abc' |
| HEX() | HEX(expr)  返回expr的16进制表示  如果expr是字符串，则返回每个字符的ASCII码的16进制表示  如果expr是数字，即为10进制转16   1. **SELECT** X'616263', HEX('abc'), UNHEX(HEX('abc')); 2. -> 'abc', 616263, 'abc' 3. **SELECT** HEX(255), CONV(HEX(255),16,10); 4. -> 'FF', 255 |
| INSERT() | INSERT(str,pos,len,newstr)  在str的pos位置开始，用newstr替换len长度的字符  pos从1开始  如果pos不在str长度范围内，则返回str原值  如果从pos开始len不在str长度范围内，则只替换newstr长度字符，并放弃其余字符  只要有一个参数为null，则返回null   1. **SELECT** **INSERT**('Quadratic', 3, 4, 'What'); 2. -> 'QuWhattic' 3. **SELECT** **INSERT**('Quadratic', -1, 4, 'What'); 4. -> 'Quadratic' 5. **SELECT** **INSERT**('Quadratic', 3, 100, 'What'); 6. -> 'QuWhat' |
| INSTR() | INSTR(str,substr)  返回substr在str中第一次出现的位置  如果没有则返回0  仅在字符编码为binary时，大小写敏感   1. **SELECT** INSTR('foobarbar', 'bar'); 2. -> 4 3. **SELECT** INSTR('xbar', 'foobar'); 4. -> 0 |
| LCASE() | 等同LOWER() |
| LEFT() | LEFT(str,len)  返回str左边指定长度len子串  如果任一参数NULL，返回null |
| LENGTH() | LENGTH(str)  返回str的字节长度 |
| LIKE | Simple pattern matching |
| LOAD\_FILE() | LOAD\_FILE(file\_name)  1返回本地服务器上完整路径file\_name下的有读权限的文件内容  2文件大小必须小于系统参数max\_allowed\_packet的值  3如果指定了系统参数secure\_file\_priv，文件必须存在于secure\_file\_priv指定的文件夹  4如果文件无法读取，则返回NULL |
| LOCATE() | LOCATE(substr,str)  LOCATE(substr,str,pos)  返回substr在str的pos位置开始第一次出现的位置  如果未指定pos，则从第一个开始  如果未发现，则返回0  仅在字符编码为binary时，大小写敏感   1. **SELECT** LOCATE('bar', 'foobarbar'); 2. -> 4 3. **SELECT** LOCATE('xbar', 'foobar'); 4. -> 0 5. **SELECT** LOCATE('bar', 'foobarbar', 5); 6. -> 7 |
| LOWER() | LOWER(str)  str字母全小写  对binary类型字符串无效   1. **SET** @str = **BINARY** 'New York'; 2. **SELECT** LOWER(@str), LOWER(CONVERT(@str USING latin1)); 3. -> New York, new york |
| LPAD() | LPAD(str,len,padstr)  如果len大于str字符长度，左边补齐padstr后，返回  如果len小于str字符长度，则返回len长度的子串   1. **SELECT** LPAD('hi',4,'??'); 2. -> '??hi' 3. **SELECT** LPAD('hi',1,'??'); 4. -> 'h' |
| LTRIM() | LTRIM(str)  移除左边空格 |
| MAKE\_SET() | MAKE\_SET(bits,str1,str2,...)  返回bits二进制位中bit=1的对应未知的字符串集合  多个用逗号隔开  如果对应位置不存在或null，则忽略   1. **SELECT** MAKE\_SET(1,'a','b','c'); 2. -> 'a' 3. --  1｜4 -> 101 4. **SELECT** MAKE\_SET(1 | 4,'hello','nice','world'); 5. -> 'hello,world' 6. **SELECT** MAKE\_SET(1 | 4,'hello','nice',NULL,'world'); 7. -> 'hello' 8. **SELECT** MAKE\_SET(0,'a','b','c'); 9. -> '' |
| MATCH | Perform full-text search |
| MID() | MID(str,pos,len)  等同SUBSTRING(str,pos,len) |
| OCT() | OCT(N)  获得N的8进制表示 |
| OCTET\_LENGTH() | 等同LENGTH() |
| ORD() | ORD(str)  返回str第一个字符的ascii值 |
| POSITION() | POSITION(substr IN str)  等同LOCATE(substr,str). |
| QUOTE() | QUOTE(str)  转义str中的单引号，防注入 |
| REPEAT() | REPEAT(str,count)  将str重复count次，并返回   1. **SELECT** REPEAT('MySQL', 3); 2. -> 'MySQLMySQLMySQL' |
| REPLACE() | REPLACE(str,from\_str,to\_str)  替换str中所有from\_str为to\_str  大小写敏感 |
| REVERSE() | REVERSE(str)  反相输出str   1. **SELECT** REVERSE('abc'); 2. -> 'cba' |
| RIGHT() | RIGHT(str,len)  从str右边开始返回len长度的子串   1. **SELECT** RIGHT('foobarbar', 4); 2. -> 'rbar' |
| RPAD() | RPAD(str,len,padstr)  如果len大于str字符长度，右边补齐padstr后，返回  如果len小于str字符长度，则返回len长度的子串   1. **SELECT** RPAD('hi',5,'?'); 2. -> 'hi???' 3. **SELECT** RPAD('hi',1,'?'); 4. -> 'h' |
| RTRIM() | RTRIM(str)  去除str右边空格 |
| SOUNDEX() | SOUNDEX(str)  将str转为描述语音表示的字母模式 |
| SOUNDS LIKE | expr1 SOUNDS LIKE expr2  等同于SOUNDEX(expr1) = SOUNDEX(expr2). |
| SPACE() | SPACE(N)  返回有N个空格的字符串 |
| SUBSTR() | SUBSTR(str,pos),  SUBSTR(str FROM pos),  SUBSTR(str,pos,len),  SUBSTR(str FROM pos FOR len)  等同SUBSTRING() |
| SUBSTRING() | SUBSTRING(str,pos),  SUBSTRING(str FROM pos),  SUBSTRING(str,pos,len),  SUBSTRING(str FROM pos FOR len)  返回str的从pos位置开始长度为len的子串  如果pos为非负数，从左到右  如果pos为负数，则从右到左  如果len小于1，则返回空字符串   1. **SELECT** SUBSTRING('Quadratically',5); 2. -> 'ratically' 3. **SELECT** SUBSTRING('foobarbar' **FROM** 4); 4. -> 'barbar' 5. **SELECT** SUBSTRING('Quadratically',5,6); 6. -> 'ratica' 7. **SELECT** SUBSTRING('Sakila', -3); 8. -> 'ila' 9. **SELECT** SUBSTRING('Sakila', -5, 3); 10. -> 'aki' 11. **SELECT** SUBSTRING('Sakila' **FROM** -4 **FOR** 2); 12. -> 'ki' |
| SUBSTRING\_INDEX() | SUBSTRING\_INDEX(str,delim,count)  返回str的第count个delim之前的子串  如果count为非负数，从左到右  如果count为负数，则从右到左   1. **SELECT** SUBSTRING\_INDEX('www.mysql.com', '.', 2); 2. -> 'www.mysql' 3. **SELECT** SUBSTRING\_INDEX('www.mysql.com', '.', -2); 4. -> 'mysql.com' |
| TO\_BASE64() | TO\_BASE64(str)  将str作base64编码 |
| TRIM() | TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str),  TRIM([remstr FROM] str)  去除str中remstr字符  默认是去除左右空格   1. **SELECT** TRIM('  bar   '); 2. -> 'bar' 3. **SELECT** TRIM(LEADING 'x' **FROM** 'xxxbarxxx'); 4. -> 'barxxx' 5. **SELECT** TRIM(BOTH 'x' **FROM** 'xxxbarxxx'); 6. -> 'bar' 7. **SELECT** TRIM(TRAILING 'xyz' **FROM** 'barxxyz'); 8. -> 'barx' |
| UCASE() | 等同UPPER() |
| UNHEX() | UNHEX(str)  将16进制的str字符串转成ASCII字符串  如果str中存在非16进制数值，则返回NULL  如果数据类型是binary，则允许str为null  等同于CONV(HEX(N),16,10) |
| UPPER() | UPPER(str)  将str字符串中的字母转为大写 |
| WEIGHT\_STRING() | Return the weight string for a string |
| REGEXP |  |
| NOT REGEXP |  |
| RLIKE |  |

## 地区列表

https://dev.mysql.com/doc/refman/5.7/en/locale-support.html

|  |  |
| --- | --- |
| **Locale Value** | **Meaning** |
| ar\_AE | Arabic - United Arab Emirates |
| ar\_BH | Arabic - Bahrain |
| ar\_DZ | Arabic - Algeria |
| ar\_EG | Arabic - Egypt |
| ar\_IN | Arabic - India |
| ar\_IQ | Arabic - Iraq |
| ar\_JO | Arabic - Jordan |
| ar\_KW | Arabic - Kuwait |
| ar\_LB | Arabic - Lebanon |
| ar\_LY | Arabic - Libya |
| ar\_MA | Arabic - Morocco |
| ar\_OM | Arabic - Oman |
| ar\_QA | Arabic - Qatar |
| ar\_SA | Arabic - Saudi Arabia |
| ar\_SD | Arabic - Sudan |
| ar\_SY | Arabic - Syria |
| ar\_TN | Arabic - Tunisia |
| ar\_YE | Arabic - Yemen |
| be\_BY | Belarusian - Belarus |
| bg\_BG | Bulgarian - Bulgaria |
| ca\_ES | Catalan - Spain |
| cs\_CZ | Czech - Czech Republic |
| da\_DK | Danish - Denmark |
| de\_AT | German - Austria |
| de\_BE | German - Belgium |
| de\_CH | German - Switzerland |
| de\_DE | German - Germany |
| de\_LU | German - Luxembourg |
| el\_GR | Greek - Greece |
| en\_AU | English - Australia |
| en\_CA | English - Canada |
| en\_GB | English - United Kingdom |
| en\_IN | English - India |
| en\_NZ | English - New Zealand |
| en\_PH | English - Philippines |
| en\_US | English - United States |
| en\_ZA | English - South Africa |
| en\_ZW | English - Zimbabwe |
| es\_AR | Spanish - Argentina |
| es\_BO | Spanish - Bolivia |
| es\_CL | Spanish - Chile |
| es\_CO | Spanish - Colombia |
| es\_CR | Spanish - Costa Rica |
| es\_DO | Spanish - Dominican Republic |
| es\_EC | Spanish - Ecuador |
| es\_ES | Spanish - Spain |
| es\_GT | Spanish - Guatemala |
| es\_HN | Spanish - Honduras |
| es\_MX | Spanish - Mexico |
| es\_NI | Spanish - Nicaragua |
| es\_PA | Spanish - Panama |
| es\_PE | Spanish - Peru |
| es\_PR | Spanish - Puerto Rico |
| es\_PY | Spanish - Paraguay |
| es\_SV | Spanish - El Salvador |
| es\_US | Spanish - United States |
| es\_UY | Spanish - Uruguay |
| es\_VE | Spanish - Venezuela |
| et\_EE | Estonian - Estonia |
| eu\_ES | Basque - Spain |
| fi\_FI | Finnish - Finland |
| fo\_FO | Faroese - Faroe Islands |
| fr\_BE | French - Belgium |
| fr\_CA | French - Canada |
| fr\_CH | French - Switzerland |
| fr\_FR | French - France |
| fr\_LU | French - Luxembourg |
| gl\_ES | Galician - Spain |
| gu\_IN | Gujarati - India |
| he\_IL | Hebrew - Israel |
| hi\_IN | Hindi - India |
| hr\_HR | Croatian - Croatia |
| hu\_HU | Hungarian - Hungary |
| id\_ID | Indonesian - Indonesia |
| is\_IS | Icelandic - Iceland |
| it\_CH | Italian - Switzerland |
| it\_IT | Italian - Italy |
| ja\_JP | Japanese - Japan |
| ko\_KR | Korean - Republic of Korea |
| lt\_LT | Lithuanian - Lithuania |
| lv\_LV | Latvian - Latvia |
| mk\_MK | Macedonian - North Macedonia |
| mn\_MN | Mongolia - Mongolian |
| ms\_MY | Malay - Malaysia |
| nb\_NO | Norwegian(Bokmål) - Norway |
| nl\_BE | Dutch - Belgium |
| nl\_NL | Dutch - The Netherlands |
| no\_NO | Norwegian - Norway |
| pl\_PL | Polish - Poland |
| pt\_BR | Portugese - Brazil |
| pt\_PT | Portugese - Portugal |
| rm\_CH | Romansh - Switzerland |
| ro\_RO | Romanian - Romania |
| ru\_RU | Russian - Russia |
| ru\_UA | Russian - Ukraine |
| sk\_SK | Slovak - Slovakia |
| sl\_SI | Slovenian - Slovenia |
| sq\_AL | Albanian - Albania |
| sr\_RS | Serbian - Serbia |
| sv\_FI | Swedish - Finland |
| sv\_SE | Swedish - Sweden |
| ta\_IN | Tamil - India |
| te\_IN | Telugu - India |
| th\_TH | Thai - Thailand |
| tr\_TR | Turkish - Turkey |
| uk\_UA | Ukrainian - Ukraine |
| ur\_PK | Urdu - Pakistan |
| vi\_VN | Vietnamese - Vietnam |
| zh\_CN | Chinese - China |
| zh\_HK | Chinese - Hong Kong |
| zh\_TW | Chinese - Taiwan |

# Aggregate Functions

https://dev.mysql.com/doc/refman/5.7/en/aggregate-functions-and-modifiers.html

|  |  |
| --- | --- |
| Name | Description |
| AVG() | AVG([DISTINCT] expr)  求字段expr的平均值  DISTINCT表示对值集合去重 |
| BIT\_AND() | BIT\_AND(expr)  对字段expr的所有值进行二进制与运算，结果为64 bit的BIGINT |
| BIT\_OR() | BIT\_OR(expr)  对字段expr的所有值进行二进制或运算，结果为64 bit的BIGINT |
| BIT\_XOR() | BIT\_XOR(expr)  对字段expr的所有值进行二进制与运算，结果为64 bit的BIGINT |
| COUNT() | COUNT(expr)  1 默认情况下统计expr字段中非NULL的数量，返回值为BIGINT类型  2 在如Innodb等事务存储引擎中，多事务情况下，count结果值可能不准确  3 COUNT(\*)，返回所有被检索到总行数，即使存在null值  3.1 在5.7.18之前，innodb引擎执行count(\*)扫描聚合索引  3.2在5.7.18，innodb引擎执行count(\*)扫描最小可用辅助索引，除非选中的辅助索引跟优化器提示的索引不一样，如果没有合适的辅助索引，才通过扫描聚合索引实现(As of MySQL 5.7.18, InnoDB processes SELECT COUNT(\*) statements by traversing the smallest available secondary index unless an index or optimizer hint directs the optimizer to use a different index. If a secondary index is not present, the clustered index is scanned.)  3.3 innodb中count(1)和count(\*)执行方式一样  3.4 MyISAM中COUNT(\*)在单表且无where情况下，最快  3.5 MyISAM中COUNT（1）仅在表的第一个字段定义为NOT NULL时才和COUNT(\*)效率一样  3.6 当执行COUNT(\*)时，索引不在缓冲池中，性能会明显下降 |
| COUNT(DISTINCT) | COUNT(DISTINCT expr,[expr...])  1 返回无重复非NULL值个数   1. **SELECT** COUNT(**DISTINCT** results) **FROM** student; |
| GROUP\_CONCAT() | 1. GROUP\_CONCAT([**DISTINCT**] expr [,expr ...] 2. [**ORDER** **BY** {unsigned\_integer | col\_name | expr} 3. [**ASC** | **DESC**] [,col\_name ...]] 4. [SEPARATOR str\_val])   1 将GroupBy查询结果字段值expr [,expr ...]，按orderBy规则排序后，用SEPARATOR串联起来，并返回  2 排序默认ASC  3 默认SEPARATOR是英文逗号  4 返回值最大长度默认1024，可根据系统参数group\_concat\_max\_len配置  5 返回值根据值长度，默认为TEXT 或 BLOB，如果group\_concat\_max\_len值小于等于512，则类型为VARCHAR 或 VARBINARY   1. **SELECT** student\_name, 2. GROUP\_CONCAT(**DISTINCT** test\_score 3. **ORDER** **BY** test\_score **DESC** SEPARATOR ' ') 4. **FROM** student 5. **GROUP** **BY** student\_name; |
| JSON\_ARRAYAGG() | JSON\_ARRAYAGG(col\_or\_expr)  从5.7.22.开始提供  将指定字段或表达式结果集封装成数组   1. **SELECT** o\_id, JSON\_ARRAYAGG(attribute) **AS** attributes 2. > **FROM** t3 **GROUP** **BY** o\_id; 3. +------+---------------------+ 4. | o\_id | attributes          | 5. +------+---------------------+ 6. |    2 | ["color", "fabric"] | 7. |    3 | ["color", "shape"]  | 8. +------+---------------------+ |
| JSON\_OBJECTAGG() | JSON\_OBJECTAGG(key, value)  将指定的字段值，组成json键值对  如果存在重复key，输出对应的value为最后一个   1. +------+-----------+-------+ 2. | o\_id | attribute | value | 3. +------+-----------+-------+ 4. |    2 | color     | red   | 5. |    2 | fabric    | silk  | 6. |    3 | color     | green | 7. |    3 | shape     | square| 8. +------+-----------+-------+ 10. **SELECT** o\_id, JSON\_OBJECTAGG(attribute, value) 11. > **FROM** t3 **GROUP** **BY** o\_id; 12. +------+---------------------------------------+ 13. | o\_id | JSON\_OBJECTAGG(attribute, value)      | 14. +------+---------------------------------------+ 15. |    2 | {"color": "red", "fabric": "silk"}    | 16. |    3 | {"color": "green", "shape": "square"} | 17. +------+---------------------------------------+ |
| MAX() | MAX([DISTINCT] expr)  获取最大值  如果expr是字符串，则比对字符串在字符集中的大小  如果expr是ENUM或SET类型，则比对的事实际值，而不是下标  如果没有值，则返回NULL |
| MIN() | MIN([DISTINCT] expr)  获取最小值  特性同MAX |
| STD() | STD(expr)  计算标准差  同STDDEV\_POP()，作为Mysql扩展提供 |
| STDDEV() | STDDEV(expr)  计算标准差  同STDDEV\_POP()，兼容Oracle |
| STDDEV\_POP() | STDDEV\_POP(expr)  计算标准差  SQL标准  如果结果集没有值，则返回NULL |
| STDDEV\_SAMP() | STDDEV\_SAMP(expr)  估算样本标准差 |
| SUM() | SUM([DISTINCT] expr)  计算expr的和  如果集合为空，返回null |
| VAR\_POP() | VAR\_POP(expr)  计算标准方差  SQL标准，同VARIANCE(expr) |
| VAR\_SAMP() | VAR\_SAMP(expr)  估算样本方差 |
| VARIANCE() | VARIANCE(expr)  计算标准方差  非SQL标准，作为Mysql扩展提供 |

# Cast Functions and Operators

https://dev.mysql.com/doc/refman/5.7/en/cast-functions.html

|  |  |
| --- | --- |
| Name | **Description** |
| BINARY | Cast a string to a binary string |
| CAST() | Cast a value as a certain type |
| CONVERT() | Cast a value as a certain type |

# XML Function

https://dev.mysql.com/doc/refman/5.7/en/xml-functions.html

|  |  |
| --- | --- |
| Name | **Description** |
| ExtractValue() | Extract a value from an XML string using XPath notation |
| UpdateXML() | Return replaced XML fragment |

# Bit Functions and Operators

https://dev.mysql.com/doc/refman/5.7/en/bit-functions.html

|  |  |
| --- | --- |
| Name | **Description** |
| & | Bitwise AND |
| >> | Right shift |
| << | Left shift |
| ^ | Bitwise XOR |
| BIT\_COUNT() | Return the number of bits that are set |
| | | Bitwise OR |
| ~ | Bitwise inversion |

# Encryption and Compression Functions

https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html

|  |  |
| --- | --- |
| Name | **Description** |
| AES\_DECRYPT() | Decrypt using AES |
| AES\_ENCRYPT() | Encrypt using AES |
| COMPRESS() | Return result as a binary string |
| MD5() | Calculate MD5 checksum |
| RANDOM\_BYTES() | Return a random byte vector |
| SHA1(), SHA() | Calculate an SHA-1 160-bit checksum |
| SHA2() | Calculate an SHA-2 checksum |
| UNCOMPRESS() | Uncompress a string compressed |
| UNCOMPRESSED\_LENGTH() | Return the length of a string before compression |
| VALIDATE\_PASSWORD\_STRENGTH() | Determine strength of password |

# Locking Functions

https://dev.mysql.com/doc/refman/5.7/en/locking-functions.html

|  |  |
| --- | --- |
| Name | **Description** |
| GET\_LOCK() | Get a named lock |
| IS\_FREE\_LOCK() | Whether the named lock is free |
| IS\_USED\_LOCK() | Whether the named lock is in use; return connection identifier if true |
| RELEASE\_ALL\_LOCKS() | Release all current named locks |
| RELEASE\_LOCK() | Release the named lock |

# Information Functions

https://dev.mysql.com/doc/refman/5.7/en/information-functions.html

|  |  |
| --- | --- |
| Name | **Description** |
| BENCHMARK() | Repeatedly execute an expression |
| CHARSET() | Return the character set of the argument |
| COERCIBILITY() | Return the collation coercibility value of the string argument |
| COLLATION() | Return the collation of the string argument |
| CONNECTION\_ID() | Return the connection ID (thread ID) for the connection |
| CURRENT\_USER(), CURRENT\_USER | The authenticated user name and host name |
| DATABASE() | Return the default (current) database name |
| FOUND\_ROWS() | For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause |
| LAST\_INSERT\_ID() | Value of the AUTOINCREMENT column for the last INSERT |
| ROW\_COUNT() | The number of rows updated |
| SCHEMA() | Synonym for DATABASE() |
| SESSION\_USER() | Synonym for USER() |
| SYSTEM\_USER() | Synonym for USER() |
| USER() | The user name and host name provided by the client |
| VERSION() | Return a string that indicates the MySQL server version |

# Miscellaneous FUnctions

|  |  |
| --- | --- |
| Name | **Description** |
| ANY\_VALUE() | Suppress ONLY\_FULL\_GROUP\_BY value rejection |
| DEFAULT() | Return the default value for a table column |
| INET\_ATON() | Return the numeric value of an IP address |
| INET\_NTOA() | Return the IP address from a numeric value |
| INET6\_ATON() | Return the numeric value of an IPv6 address |
| INET6\_NTOA() | Return the IPv6 address from a numeric value |
| IS\_IPV4() | Whether argument is an IPv4 address |
| IS\_IPV4\_COMPAT() | Whether argument is an IPv4-compatible address |
| IS\_IPV4\_MAPPED() | Whether argument is an IPv4-mapped address |
| IS\_IPV6() | Whether argument is an IPv6 address |
| MASTER\_POS\_WAIT() | Block until the slave has read and applied all updates up to the specified position |
| NAME\_CONST() | Cause the column to have the given name |
| SLEEP() | Sleep for a number of seconds |
| UUID() | Return a Universal Unique Identifier (UUID) |
| UUID\_SHORT() | Return an integer-valued universal identifier |
| VALUES() | Define the values to be used during an INSERT |