# 05 很有用的条件判断函数与系统函数

更新时间：2020-03-11 09:45:06

> 人生太短，要干的事太多，我要争分夺秒。——爱迪生

高级语言中对于条件判断我们可以使用类似 if…else、switch…case 等语句，它们非常方便实用。为了避免手动多次转换与条件匹配，MySQL 同样提供了功能强大的条件函数。MySQL 中的系统信息包含：数据库的版本号、当前登录用户、当前连接数、系统字符集等等。这些信息在特定的环境中非常有用，例如：查看连接数定位 MySQL 是否过载、查看字符集定位乱码原因等等。这一节里，我们就来看一看 MySQL 中有哪些条件判断和系统函数，以及它们的使用方法。

## 1 常用的条件判断函数

我们可以把常用的条件判断函数分为两类：IF 和 CASE。其中，IF 又包含 IF、IFNULL 和 NULLIF，它们都是单一的条件比对。如果想要实现多条件比对，则需要使用 CASE 语句。在讲解这两类函数之前，我们假定数据表 worker 中存储了如下的数据。

```
mysql> SELECT id, type, name, salary FROM worker;
+----+------+--------+--------+
| id | type | name   | salary |
+----+------+--------+--------+
|  1 | A    | tom    |   1800 |
|  2 | B    | jack   |   2100 |
|  3 | C    | pony   |   NULL |
|  4 | B    | tony   |   3600 |
|  5 | B    | marry  |   1900 |
|  6 | C    | tack   |   1200 |
|  7 | A    | tick   |   NULL |
|  8 | B    | clock  |   2000 |
|  9 | C    | noah   |   1500 |
| 10 | C    | jarvis |   1800 |
+----+------+--------+--------+
```

如果不做特别说明，接下来的查询示例中都会使用到 worker 表及其当前存储的数据。

#### 1.1 IF 条件判断函数

首先，我们来看一看 IF (expr, v1, v2) 函数。expr 是表达式的意思，它的含义是：如果 expr 为真（expr<>0 and expr<>NULL），则 IF 函数的返回值是 v1，否则，返回 v2。IF 函数的返回值是数字还是字符串，则视其所在语境而定。

如果你没有用过 IF，可能通过我的解释还是不能很好的理解，那么，看个例子吧。worker 表中有一列 type，用于标识员工的工作岗位（暂且不用关心具体指代什么），如果我想区分出来，且给出对应的标识，可以这样做。

```
mysql> SELECT name, IF(type='A', '研发', '非研发') AS type FROM worker WHERE id IN (1, 2);
+------+-----------+
| name | type      |
+------+-----------+
| tom  | 研发      |
| jack | 非研发    |
+------+-----------+
```

从结果中可以得出结论，SQL 中的 IF 函数与高级语言中的 if…else 功能相同。另外，可以注意到，不知道什么原因，部分员工的 salary 是 NULL，这是没有意义的。如果把 NULL 显示为 0 是不是更好一些呢 ？此时，IFNULL (v1, v2) 函数可以解决这个问题。

IFNULL (v1, v2) 表达的语义是：如果 v1 不为 NULL，则返回 v1，否则，返回 v2。同样，IFNULL 函数的返回值是数字还是字符串，则视其所在语境而定。我们可以利用 IFNULL 将 salary 是 NULL 的列值变成 0。

```
mysql> SELECT name, IFNULL(salary, 0) AS salary FROM worker WHERE id IN (1, 2, 3);
+------+--------+
| name | salary |
+------+--------+
| tom  |   1800 |
| jack |   2100 |
| pony |      0 |
+------+--------+
```

与 IFNULL 的名称很相似的一个函数是 NULLIF (v1, v2)，它表达的语义是：如果 v1 等于 v2，那么返回值是 NULL，否则返回值为 v1。下面，我们来看一个 “没有太多意义” 的例子。

```
mysql> SELECT id, name, NULLIF(salary, NULL) AS salary FROM worker WHERE id IN (1, 2, 3);
+----+-------+--------+
| id | name | salary |
+----+-------+--------+
|  1 | tom  |   1800 |
|  2 | jack |   2100 |
|  3 | pony |   NULL |
+----+-------+--------+
```

可以看到，id 为 1 和 2 的记录，salary 不为 NULL，直接返回了 salary。而 id 为 3 的记录，salary 等于 NULL，返回了 NULL。

#### 1.2 CASE 条件判断函数

IF 条件判断函数所表达的语义是非 A 即 B，也就是单个条件的判断。我们接下来要看到的 CASE 函数则能够实现多条件的匹配。首先，我们先来看一看 CASE 函数的语法：

```
CASE expr
 WHEN v1 THEN r1
 ......
 WHEN vx THEN rx
 ELSE rn
END
```

我们通常把 CASE 函数叫做 “CASE WHEN THEN”，聪明的你一定知道这是为什么。SQL 语句中的 CASE 语句是标准的 SQL 语法，适用于一个条件判断有多种可能值的情况下分别去执行不同的操作，或返回不同的结果值。

CASE 函数有两种写法：简单 CASE 函数写法、CASE 搜索函数写法，它们的区别在于：

- 简单 CASE 函数写法只适合相等条件判断，不能用于大于、小于、不等于的判断
- CASE 搜索函数写法适合复杂条件判断，可用于大于、小于、不等于的判断

理解知识点的最好方式就是学习示例，并模仿示例做实践。首先，我们先来看一个简单 CASE 函数写法的例子（为了更清晰的看到 SQL 语法，做了格式化处理）：

```
mysql> SELECT
    ->     name,
    ->     (CASE type
    ->       WHEN 'A' THEN '研发'
    ->       WHEN 'B' THEN '测试'
    ->       WHEN 'C' THEN '运维'
    ->       ELSE '其他'
    ->     END) AS type
    ->   FROM
    ->     worker
    ->   WHERE
    ->     id IN (
    ->       1, 2, 3
    ->     );
+------+--------+
| name | type   |
+------+--------+
| tom  | 研发   |
| jack | 测试   |
| pony | 运维   |
+------+--------+
```

这条 SQL 语句比较简单，非常容易看懂，就是根据 type 的值返回不同的值，写法和功能都类似于高级语言中的 switch…case 语句。需要特别注意的是 type（CASE 后面的） 在 SQL 语句中的位置。

下面，我们使用 CASE 搜索函数的写法来实现同样的查询：

```
mysql> SELECT
    ->     name,
    ->     (CASE
    ->       WHEN type='A' THEN '研发'
    ->       WHEN type='B' THEN '测试'
    ->       WHEN type='C' THEN '运维'
    ->       ELSE '其他'
    ->     END) AS type
    ->   FROM
    ->     worker
    ->   WHERE
    ->     id IN (
    ->       1, 2, 3
    ->     );
+------+--------+
| name | type   |
+------+--------+
| tom  | 研发   |
| jack | 测试   |
| pony | 运维   |
+------+--------+
```

可以清晰的看到，type 的位置从 CASE 的后面换到了 WHEN 的后面。同时，也就理解了我刚刚所说的这两种写法之间的区别。下面，我们再来看一个例子：

```
mysql> SELECT
    ->     name,
    ->     (CASE
    ->       WHEN salary >= 2000 THEN '高收入'
    ->       WHEN salary <= 1500 THEN '低收入'
    ->       ELSE '中等收入'
    ->     END) AS salary
    ->   FROM
    ->     worker
    ->   WHERE
    ->     id IN (
    ->       1, 2, 9
    ->     );
+------+--------------+
| name | salary       |
+------+--------------+
| tom  | 中等收入     |
| jack | 高收入       |
| noah | 低收入       |
+------+--------------+
```

对于涉及数值范围判断等等类似的例子，则只能使用 CASE 搜索函数的写法。

讲解了 IF 与 CASE 的功能与语法之后，我们来对这两类条件判断函数做一个总结：

- CASE 是 SQL 标准定义的，而 IF 是数据库系统的扩展
- 在高级语言中，CASE 可以使用 IF 来代替，但是 SQL 中却不行
- 在 SQL 的存储过程和触发器中，用 IF 替代 CASE 的代价是非常高的，难以应用

CASE 语句可以让 SQL 变得简单高效，提高执行效率，且通常不会引起性能问题，所以，通常应该作为首选。最后，理解和掌握知识点的最好方式就是去照猫画虎，模仿别人的写法去实现自己想要的功能。现在就打开你的数据库，试一试这些 SQL 查询吧。

# 2 常用的系统函数

MySQL 提供的系统函数虽然功能强大，但是在学习和使用上都是非常简单的。这里我将这些系统函数分为三类进行讲解：MySQL 自身的基本信息、当前用户信息、库和表相关的信息。

### 2.1 MySQL 自身的基本信息

MySQL 自身的基本信息大多都存储在系统表中，只提供了一个系统函数：VERSION ()。VERSION () 函数返回的是 UTF-8 字符集编码的字符串，标识当前登录的 MySQL 服务器版本。

```
mysql> SELECT VERSION();
+-----------+
| VERSION() |
+-----------+
| 5.7.28    |
+-----------+
```

### 2.2 当前用户信息

用户信息包含了客户端连接 ID、用户名以及当前选择的数据库（由于当前选择的数据库是用户行为，所以，我这里把它也归类为用户信息）。首先，我们来看一看怎样查询客户端连接 ID：

```
mysql> SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|               4 |
+-----------------+
```

对于已经建立连接的客户端，MySQL 都会用一个唯一的 ID 去标识它，而 CONNECTION_ID () 函数则可以打印这个 ID。那么，既然 MySQL 可以打印它，就一定会有地方存储它（不仅是对于当前信息，对于其他信息大多也是成立的）。

这个连接 ID 实际存储于 MySQL 的两张系统表中：information_schema.PROCESSLIST（ID 字段值）、performance_schema.threads（PROCESSLIST_ID 字段值）。如下所示：

```
mysql> SELECT * FROM information_schema.PROCESSLIST;
+----+------+-----------------+--------------------+---------+------+-----------+-----------------------------------------------------------+
| ID | USER | HOST            | DB                 | COMMAND | TIME | STATE     | INFO                                                      |
+----+------+-----------------+--------------------+---------+------+-----------+-----------------------------------------------------------+
|  4 | root | localhost       | imooc_mysql        | Query   |    0 | executing | SELECT * FROM information_schema.PROCESSLIST |
|  2 | root | localhost:50675 | imooc_mysql        | Sleep   |   27 |           | NULL                                                      |
|  5 | root | localhost:58433 | information_schema | Sleep   |   67 |           | NULL                                                      |
|  3 | root | localhost:50676 | NULL               | Sleep   |   26 |           | NULL                                                      |
|  6 | root | localhost:58446 | NULL               | Sleep   |   38 |           | NULL                                                      |
+----+------+-----------------+--------------------+---------+------+-----------+-----------------------------------------------------------+

mysql> SELECT THREAD_ID, NAME, TYPE, PROCESSLIST_ID FROM performance_schema.threads WHERE PROCESSLIST_ID = 4;
+-----------+----------------------------+------------+----------------+
| THREAD_ID | NAME                       | TYPE       | PROCESSLIST_ID |
+-----------+----------------------------+------------+----------------+
|        30 | thread/sql/one_connection | FOREGROUND |              4 |
+-----------+----------------------------+------------+----------------+
```

另外，通过 SHOW PROCESSLIST 和 SHOW FULL PROCESSLIST 语句也可以查看连接信息。如果你当前使用的是 root 账户登录，可以查看到所有的用户连接。如果是普通账户，则只能看到自己的。SHOW PROCESSLIST 只会打印前 100 条连接信息，而 SHOW FULL PROCESSLIST 正如它的名字一样，可以打印全部的连接信息。例如：

```
mysql> SHOW PROCESSLIST;
+----+------+-----------------+--------------------+---------+------+----------+------------------+
| Id | User | Host            | db                 | Command | Time | State    | Info             |
+----+------+-----------------+--------------------+---------+------+----------+------------------+
|  2 | root | localhost:50675 | imooc_mysql        | Sleep   |   61 |          | NULL             |
|  3 | root | localhost:50676 | NULL               | Sleep   |   42 |          | NULL             |
|  4 | root | localhost       | imooc_mysql        | Query   |    0 | starting | SHOW PROCESSLIST |
|  5 | root | localhost:58433 | information_schema | Sleep   |   35 |          | NULL             |
|  6 | root | localhost:58446 | NULL               | Sleep   |   14 |          | NULL             |
+----+------+-----------------+--------------------+---------+------+----------+------------------+
```

聪明的你一定可以发现这条语句打印的信息与 information_schema.PROCESSLIST 表中的列值是一致的。那么，我就来解释下这些字段代表了什么：

- Id：用户客户端连接 MySQL 时，系统自动分配的连接 ID
- User：当前连接的用户名
- Host：当前用户的 id 和 端口号
- db：当前连接选择的数据库，如果没有选择，则是 NULL
- Command：当前连接执行的命令，取值为 Sleep（睡眠）、Query（查询）、Connect（连接）
- Time：状态持续的时间，单位是秒
- State：当前连接执行 SQL 语句的状态
- Info：显示当前执行的 SQL 语句

MySQL 提供了多个系统函数（USER (), CURRENT_USER (), SYSTEM_USER (), SESSION_USER ()）用于查看用户名和主机名，通常，这些函数的返回值都是相同的。例如：

```
mysql> SELECT USER(), CURRENT_USER(), SYSTEM_USER(), SESSION_USER();
+----------------+----------------+----------------+----------------+
| USER()         | CURRENT_USER() | SYSTEM_USER()  | SESSION_USER() |
+----------------+----------------+----------------+----------------+
| root@localhost | root@localhost | root@localhost | root@localhost |
+----------------+----------------+----------------+----------------+
```

查看当前选择的数据库是非常常见的需求，MySQL 为此提供了两个功能相同的系统函数：DATABASE () 和 SCHEMA ()，这两个函数会打印 UTF-8 字符集编码的数据库名。需要注意的是，如果用户未选择数据库，例如刚刚登录时，则会打印 NULL。下面，同样给出示例：

```
-- 未选择数据库
mysql> SELECT DATABASE(), SCHEMA();
+------------+----------+
| DATABASE() | SCHEMA() |
+------------+----------+
| NULL       | NULL     |
+------------+----------+

-- 切换到 imooc_mysql
mysql> use imooc_mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

mysql> SELECT DATABASE(), SCHEMA();
+-------------+-------------+
| DATABASE()  | SCHEMA()    |
+-------------+-------------+
| imooc_mysql | imooc_mysql |
+-------------+-------------+
```

MySQL 系统函数提供的用户信息非常有用，例如利用连接 ID 标识每一次查询，可以方便的对每个用户回溯他的查询历史；可以查看当前连接的所有客户端，定位服务器超载问题等等。

### 2.3 库和表相关的信息

这里所要讲解的库和表相关的信息位于应用层面，涉及字符集、字符排列方式和自增 ID。字符集指的是系统对字符编码的方式，我们想要查看当前 MySQL 使用的字符集可以使用 CHARSET () 函数：

```
mysql> SELECT CHARSET('慕课网'), CHARSET('MySQL');
+----------------------+------------------+
| CHARSET('慕课网')    | CHARSET('MySQL') |
+----------------------+------------------+
| utf8                 | utf8             |
+----------------------+------------------+
```

当然，如果在你的机器上 CHARSET () 函数返回的字符集与我的不一致也是正常的，因为字符集是可以由你来指定的。既然涉及到字符集、字符串，就一定会有字符串的排列方式，且使用不同的字符集时，字符串的排列方式也不一样。我们可以使用 COLLATION () 系统函数来看一看字符串的排列方式：

```
mysql> select COLLATION('MySQL'), COLLATION(CONVERT('MySQL' USING gbk));
+--------------------+----------------------------------------+
| COLLATION('MySQL') | COLLATION(CONVERT('MySQL' USING gbk)) |
+--------------------+----------------------------------------+
| utf8_general_ci    | gbk_chinese_ci                         |
+--------------------+----------------------------------------+
```

可以看到，对应于 utf8 字符集，排列方式是 utf8_general_ci；而使用 gbk 字符集，排列方式变成了 gbk_chinese_ci。这其中，ci 是 case insensitive 的意思，即在比较的时候不区分大小写。

MySQL 提供了一个系统函数叫做 LAST_INSERT_ID ()，它可以返回最后生成的 AUTO_INCREMENT 值。它所表达的语义是：返回最后一个 INSERT 或 UPDATE 为 AUTO_INCREMENT 列设置的第一个生成值。说实话，看到这句描述，我也是很懵，那就直接看例子吧：

```
mysql> INSERT
    -> INTO
    ->  `worker` (
    ->  `type`, `name`, `salary`
    ->  )
    -> VALUES
    ->  ('A', 'test-1', 1000);

mysql> SELECT id FROM worker WHERE name = 'test-1';
+----+
| id |
+----+
| 11 |
+----+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|               11 |
+------------------+
```

在没有执行 INSERT 之前，当前数据表的最后一条记录的 id 是 10。所以，INSERT 对应的记录 id 是 11，而正如之前所介绍的， LAST_INSERT_ID () 函数返回了 11。这种情况是插入了一条数据，如果一个 INSERT 插入多条数据呢 ？

```
mysql> INSERT
    -> INTO
    ->  `worker` (
    ->  `type`, `name`, `salary`
    ->  )
    -> VALUES
    ->  ('A', 'test-2', 1000),
    ->  ('B', 'test-3', 1000),
    ->  ('C', 'test-4', 1000);

mysql> SELECT id FROM worker WHERE name in ('test-2', 'test-3', 'test-4');
+----+
| id |
+----+
| 12 |
| 13 |
| 14 |
+----+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|               12 |
+------------------+
```

可以看到，LAST_INSERT_ID () 函数返回值是 12，这也就是之前所说的 “为 AUTO_INCREMENT 列设置的第一个生成值”。最后，需要知道，LAST_INSERT_ID 与表是无关的，如果先把数据插入到 A 表中，再把数据插入到 B 表中，返回的是 B 表中的 id 值。

## 3 总结

合理的使用条件判断函数可以在很大程度上简化代码的编写，同时由于条件判断通常不会很复杂，所以，不用过分担心服务器的性能问题。系统函数常常用来查看系统信息与定位问题，相对来说，使用场景没有条件判断函数广泛。但是，也仍然非常重要，需要多去理解和动手实践。

## 4 问题

举例说明你在日常的工作、学习中是怎样使用条件判断函数的？

MySQL 的默认字符排列方式是大小写不敏感的，如果想要大小写敏感，你会怎么做呢？

## 5 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：INFORMATION_SCHEMA PROCESSLIST Table

MySQL 官方文档：charset

}

04 学会聚合与分组聚合是很有必要的

06 常常被忽略的用户与权限