

【MySQL】存储过程(十一)

🚗 MySQL学习·第十一站~

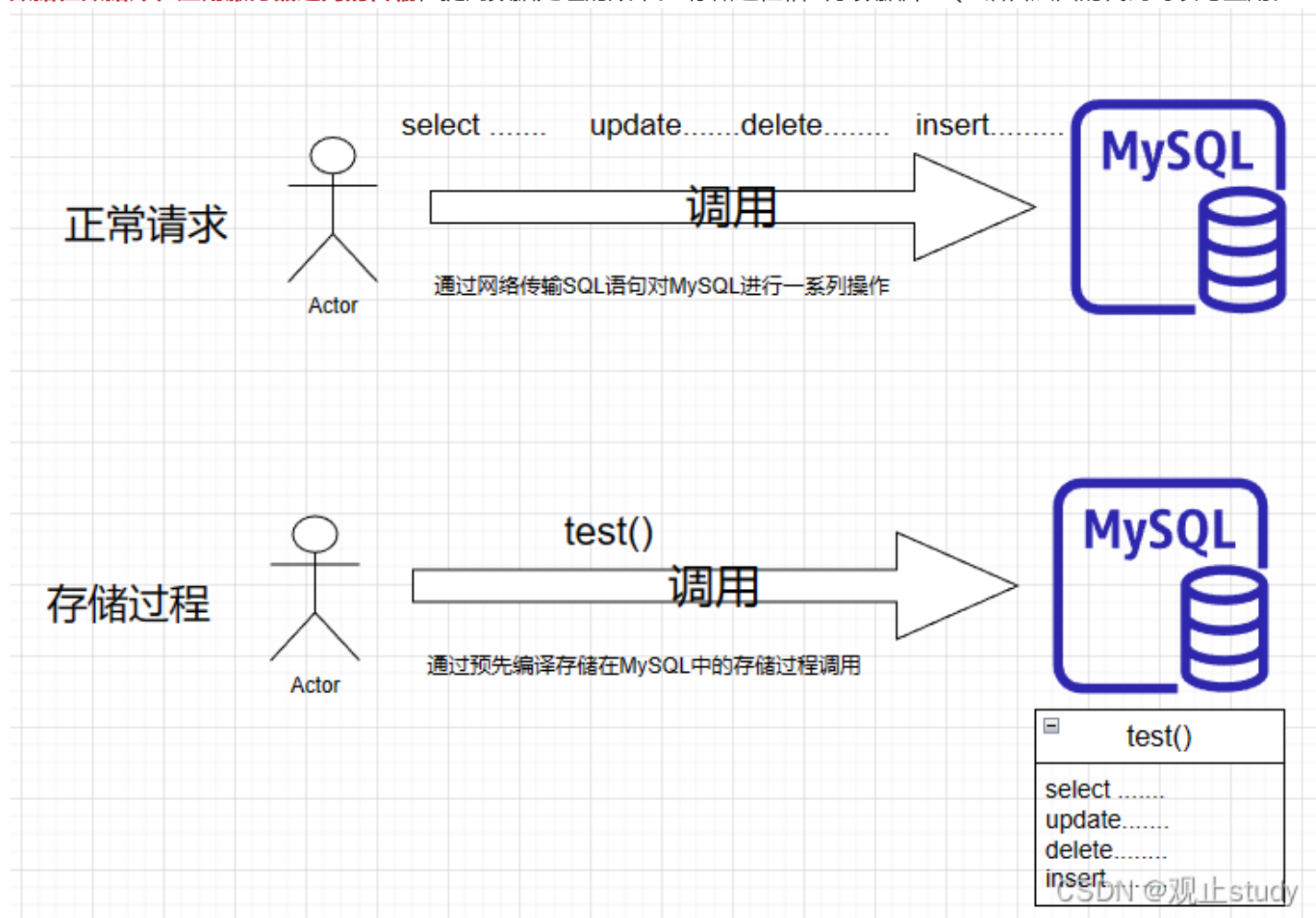
📌 本文已收录至专栏：[MySQL通关路](#)

❤️ 文末附全文思维导图，感谢各位点赞收藏支持~

🌟 学习汇总贴，超详细思维导图：[【MySQL】学习汇总\(完整思维导图\)](#)

一.引入

存储过程是**事先经过编译并存储在数据库中的一段 SQL 语句的集合**，调用存储过程可以简化应用开发人员的工作，**可以减少数据在数据库和应用服务器之间的传输**，提高数据处理的效率。存储过程相当于数据库 SQL 语言层面的代码封装与重用。



用途：

- 可以把某一业务SQL封装在存储过程中，需要用到的时候直接调用即可。
- 类似于其他语言的函数(方法)，在使用存储过程中，可以传递参数，也可以接收返回值。
- 减少客户端与数据库的网络交互，提高执行效率，如果涉及到多条SQL执行，每执行一次都是一次网络传输。而如果封装在存储过程中，我们只需要网络交互一次可能就可以了。

二.基础语法

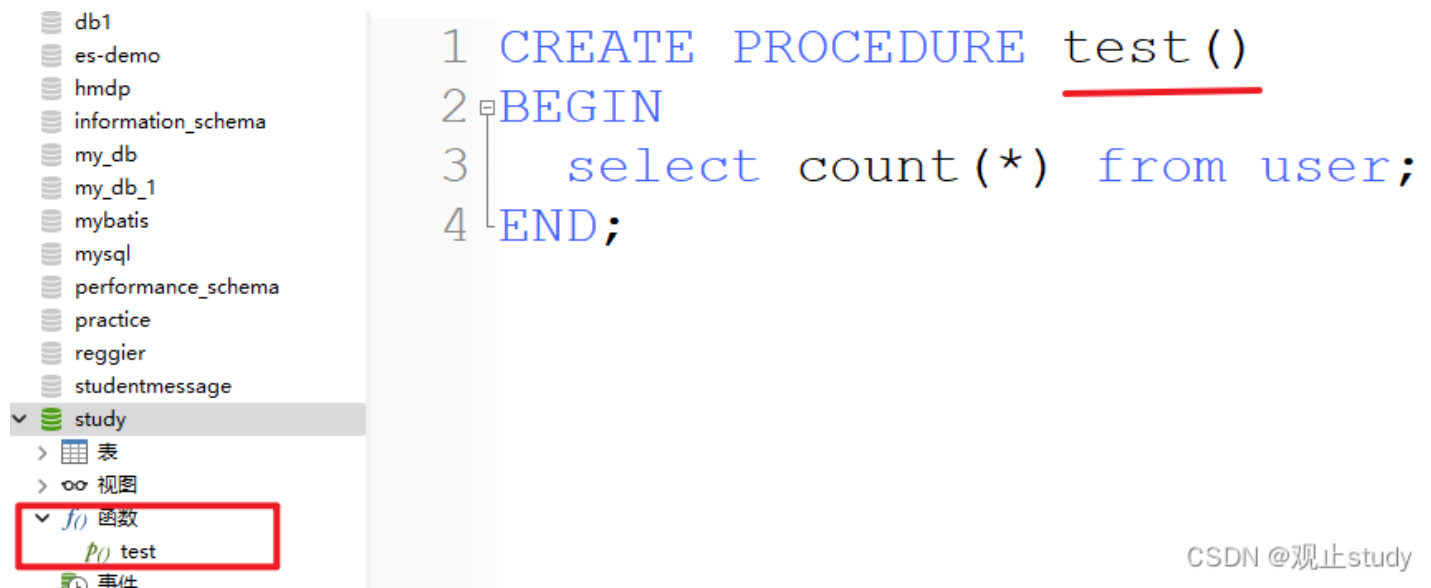
(1) 创建

(1.1) 在客户端创建

- 语法

```
CREATE PROCEDURE 存储过程名称 ([ 参数列表 ])  
BEGIN  
-- SQL语句  
END;
```

- 创建一个最简单的存储过程



```
1 CREATE PROCEDURE test()  
2 BEGIN  
3     select count(*) from user;  
4 END;
```

CSDN @观止study

(1.2) 在命令行创建

在命令行中，默认以分号表示语句的结束，因此我们无法像上述一样直接创建存储过程。

```
mysql> CREATE PROCEDURE test()  
-> BEGIN  
-> select count(*) from user;  
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 3  
mysql> END;  
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'END' at line 1  
mysql> |
```

两个分号，两个报错，系统当成了两条语句来执行

CSDN @观止study

我们需要先通过关键字 `delimiter` 重新指定SQL语句的结束符,再执行即可。

```
mysql> delimiter !! ← 可随意重新指定结束符
mysql> CREATE PROCEDURE test()
  -> BEGIN
  -> select count(*) from user;
  -> END;
  -> !!

ERROR 1304 (42000): PROCEDURE test already exists
mysql> delimiter !!
mysql> CREATE PROCEDURE test()
  -> BEGIN
  -> select count(*) from user;
  -> END;
  -> !! ←

Query OK, 0 rows affected (0.00 sec)

mysql> call test()!! ←
+-----+
| count(*) |
+-----+
|      15 |
+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

之后系统以重新指定的结束符
作为结束标识

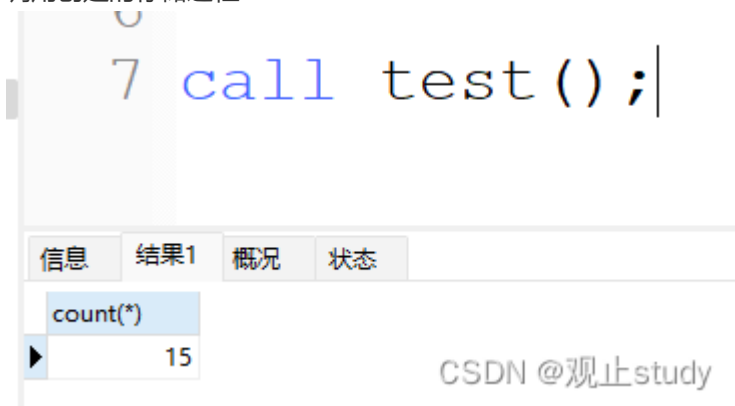
CSDN @观止study

(2) 调用

- 语法

CALL 名称 ([参数列表]);

- 调用创建的存储过程



(3) 查看

- 语法

-- 方式一: 查询指定数据库的存储过程及状态信息

SELECT * **FROM** INFORMATION_SCHEMA.ROUTINES **WHERE** ROUTINE_SCHEMA = '数据库名称';

-- 方式二: 查询某个存储过程的定义

SHOW CREATE PROCEDURE 存储过程名称;

- 方式一

```
10 SELECT * FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_SCHEMA = 'study';
11
```

信息	结果1	概况	状态			
SPECIFIC_NAME	ROUTINE_CATALOG	ROUTINE_SCHEMA	ROUTINE_NAME	ROUTINE_TYPE	ROUTINE_DEFINITION	DATA_TYPE
test	def	study	test	PROCEDURE	BEGINselect count(*) from user;END	

CSDN @观止study

- 方式二

```
13 SHOW CREATE PROCEDURE test;
```

信息	结果1	概况	状态
Procedure	sql_mode	Create Procedure	
test	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION	CREATE DEFINER='root'@'localhost' PROCEDURE `test`()BEGINselect count(*) from user;END	

(4) 删除

- 语法

```
DROP PROCEDURE [ IF EXISTS ] 存储过程名称;
```

三.进阶使用

由于我们可以在存储过程中封装多条SQL，传递变量，获取返回结果。因此面对复杂的情况，我们可以辅以一系列的逻辑。

(1) 变量

在MySQL中变量分为三种类型: 系统变量、用户定义变量、局部变量。

(1.1) 系统变量

系统变量 是MySQL服务器提供，属于服务器层面，其中又分为全局变量、会话变量

- 全局变量(**GLOBAL**): 设置后针对于所有的会话生效
- 会话变量(**SESSION**): 只对当前会话生效，在另外一个会话窗口就不生效了

(1.1.1) 查看系统变量

- 方式一：查看所有系统变量

```
SHOW [ SESSION | GLOBAL ] VARIABLES;
```

```
1 SHOW GLOBAL VARIABLES;
```

信息	结果1	概况	状态
Variable_name Value			
	▶ activate_all_roles_on_login	OFF	
	admin_address		
	admin_port	33062	
	admin_ssl_ca		
	admin_ssl_cacpath		
	admin_ssl_cert		
	admin_ssl_cipher		
	admin_ssl_crl		
	admin_ssl_crlpath		
	admin_ssl_key		
	admin_tls_ciphersuites		
	admin_tls_version	TLSv1.2,TLSv1.3	
	authentication_policy	*,,	
	auto_generate_certs	ON	
	auto_increment_increment	1	
	auto_increment_offset	1	
	autocommit	ON	
	automatic_sp_privileges	ON	

CSDN @观止study

- 方式二：通过LIKE模糊匹配方式查找变量

```
SHOW [ SESSION | GLOBAL ] VARIABLES LIKE '.....';
```

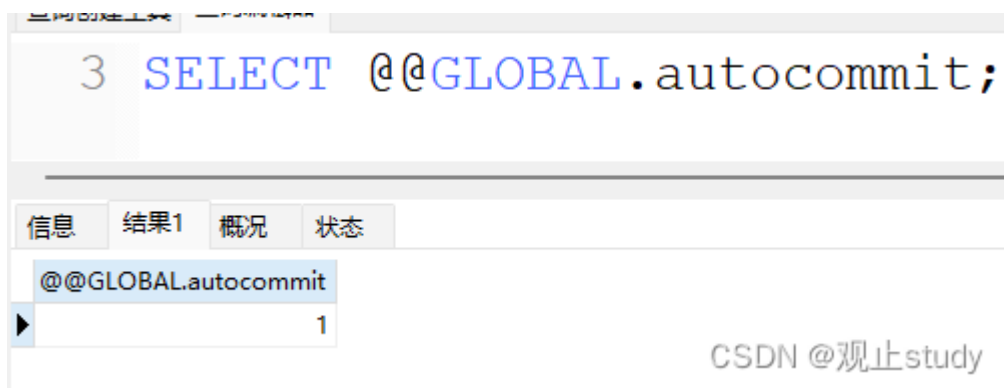
```
1 SHOW GLOBAL VARIABLES LIKE 'auto%'
```

信息	结果1	概况	状态
Variable_name Value			
	auto_generate_certs	ON	
	auto_increment_increment	1	
	auto_increment_offset	1	
	autocommit	ON	
	automatic_sp_privileges	ON	

CSDN @观止study

- 方式三：查看指定变量的值

```
SELECT @@[SESSION | GLOBAL] 系统变量名;
```



(1.1.2) 设置系统变量

- 语法一

`SET [SESSION | GLOBAL] 系统变量名 = 值 ;`



- 语法二

`SET @@[SESSION | GLOBAL]系统变量名 = 值 ;`



注意事项

- 如果没有指定 `SESSION/GLOBAL`，默认是 `SESSION`，会话变量
- mysql服务重新启动之后，所设置的全局参数会失效，要想不失效，可以在 `/etc/my.cnf` 中配置

(1.2) 用户定义变量

用户根据需要自己定义的变量，可以不提前声明直接使用（返回null），在用的时候直接用"@变量名"就可以。其作用域为当前连接。

(1.2.1) 赋值（声明）

- 方式一

```
-- 赋值时，可以使用 = ，也可以使用 :=
SET @var_name = expr [, @var_name = expr] ... ;
SET @var_name := expr [, @var_name := expr] ... ;

-- ===使用示例===
set @test1 := 111 -- 可为单个变量声明赋值
set @test2 := 222,@test3=333 -- 可为多个变量声明赋值
```

- 方式二

```
SELECT @var_name := expr [, @var_name := expr] ...

SELECT 字段名 INTO @var_name FROM 表名;

-- ===使用示例===
SELECT @test4 := 444
SELECT age INTO @test5 FROM user WHERE id =1; -- 需确保结果为一个
```

```
1 SELECT age INTO @test5 FROM user WHERE id =1;
```

信息概况状态

赋值成功

[SQL]SELECT age INTO @test5 FROM user ;
[Err] 1172 - Result consisted of more than one row
赋值失败，必须确保结果为一个值
CSDN @观止study

[SQL]SELECT age INTO @test5 FROM user WHERE id =1;
受影响的行: 1
时间: 0.000s

(1.2.2) 查看

- 语法

```
SELECT @变量名 ;
```

1 SELECT @test5;

已声明赋值变量

信息结果1概况状态

@test5

23

1 SELECT @gz;

未声明赋值变量

信息结果1概况状态

@gz

(Null)

CSDN @观止study

(1.3) 局部变量

定义在局部生效的变量，访问之前，需要先使用 `DECLARE` 声明。**可用作存储过程内的局部变量和输入参数**，局部变量的**范围是在其内声明的** `BEGIN ... END` 块。

(1.3.1) 声明

- 语法

```
create procedure 存储过程名称 ([ 参数列表 ])
begin
    declare 变量名 变量类型 [DEFAULT ... ];
    -- ....
    -- SQL语句
end;
```

可以同时定义多个局部变量，变量类型就是数据库字段类型：INT、BIGINT、CHAR、VARCHAR、DATE、TIME等。

(1.3.2) 赋值

- 语法（与上述类似）

```
-- 方式一
SET 变量名 = 值 ;
-- 方式二
SET 变量名 := 值 ;
-- 方式三
SELECT 字段名 INTO 变量名 FROM 表名 ... ;
```

- 完整示例

```
1  -- 创建存储过程
2  create procedure test()
3  begin
4      -- 声明变量
5      declare user_count int default 0;
6      -- 赋值
7      select count(*) into user_count from user;
8      -- 查看变量
9      select user_count;
10 end;
11
12 -- 调用存储过程
13 call test();
```

信息	结果1	概况	状态
----	-----	----	----

user_count

15

CSDN @观止study

(2) if 判断

- 语法

```
-- 完整版
IF 条件1 THEN
    .....
ELSEIF 条件2 THEN      -- 可选
    .....
ELSE                    -- 可选
    .....
END IF;

-- 简洁版
IF 条件1 THEN
    .....
END IF;
```

在if条件判断的结构中，`ELSE IF` 结构可以有多个，也可以没有。`ELSE` 结构可以有，也可以没有。

- 使用示例

根据定义的分數score变量，判定当前分数对应的分数等级。score >= 85分，等级为优秀。score >= 60分 且 score < 85分，等级为及格。score < 60分，等级为不及格。

```
-- 创建存储过程
create procedure p3()
begin
    -- 定义变量
    declare score int default 58;
    declare result varchar(10);
    -- if判断
    if score >= 85 then
        set result := '优秀';
    elseif score >= 60 then
        set result := '及格';
    else
        set result := '不及格';
    end if;
    -- 查看参数值
    select result;
end;
-- 调用存储过程
call p3();
```

信息	结果1	概况	状态
	result		
	不及格		

上述的需求我们虽然已经实现了，但是也存在一些问题，比如：score 分数我们是在存储过程中定义死的，而且最终计算出来的分数等级，我们也仅仅是最终查询展示出来而已，非常的鸡肋。接下来我们就学习一下参数的使用，把score分数动态的传递进来，再把计算出来的分数等级作为返回值返回。

(3) 参数

参数的类型，主要分为以下三种：IN、OUT、INOUT。具体的含义如下：

类型	含义
IN	表示输入参数，也就是调用时需要传入值
OUT	表示输出参数，也就是该参数可以作为返回值
INOUT	既可以作为输入参数，也可以作为输出参数

- 语法

```
CREATE PROCEDURE 存储过程名称 ([ [IN | OUT | INOUT] 参数名 参数类型 ])
BEGIN
    -- SQL语句
END ;
```

改进一下上述if判断中查看分数等级示例

```
-- 创建存储过程
-- in score int 表示需要传入一个整数型的参数
-- out result varchar(10) 表示返回一个varchar(10)的字符串参数
create procedure p4(in score int, out result varchar(10))
begin
    if score >= 85 then
        set result := '优秀';
    elseif score >= 60 then
        set result := '及格';
    else
        set result := '不及格';
    end if;
end;

-- 定义用户变量 @result来接收返回的数据，用户变量可以不用声明
-- 调用存储过程
call p4(18, @result);
-- 查看变量值
select @result;
```

信息	结果1	概况	状态
	result		
	不及格		

结果与上述一致，但是变得更加可控。

(4) case 流程控制

- 语法一

```
CASE case_value
WHEN when_value1 THEN statement_list1
[ WHEN when_value2 THEN statement_list2] ...
[ ELSE statement_list ]
END CASE;
```

当case_value的值为 when_value1时，执行statement_list1，当值为 when_value2时，执行statement_list2，否则就执行statement_list。

- 使用示例

```
1  -- 创建存储过程
2  create procedure p7(in month int)
3  begin
4      declare result varchar(10);
5      case month
6          -- 条件1
7          when 1 then
8              set result := '第一季度';
9          -- 条件2
10         when 2 then
11             set result := '第二季度';
12         -- 条件3
13         when 3 then
14             set result := '第三季度';
15         -- 条件4
16         when 4 then
17             set result := '第四季度';
18         -- 上述都不满足
19         else
20             set result := '非法参数';
21     end case ;
22     select concat('您输入的月份为: ',month, ', 所属的季度为: ',result);
23 end;
24 -- 调用存储过程并传入参数
25 call p7(2);
```

信息	结果1	概况	状态
----	-----	----	----

concat('您输入的月份为: ',month, ', 所属的季度为: ',result)			
--	--	--	--

▶ 您输入的月份为: 2, 所属的季度为: 第二季度			
----------------------------	--	--	--

CSDN @观止study

- 语法二

```
CASE
WHEN search_condition1 THEN statement_list1
[WHEN search_condition2 THEN statement_list2] ...
[ELSE statement_list]
END CASE;
```

当条件search_condition1成立时, 执行statement_list1, 当条件search_condition2成立时, 执行statement_list2, 否则就执行 statement_list。如果判定条件有多个, 多个条件之间, 可以使用 and 或 or 进行连接。

- 使用示例

```

1  -- 创建存储过程
2  create procedure p6(in month int)
3  begin
4      declare result varchar(10);
5      case
6          -- 条件1
7          when month >= 1 and month <= 3 then
8              set result := '第一季度';
9          -- 条件2
10         when month >= 4 and month <= 6 then
11             set result := '第二季度';
12         -- 条件3
13         when month >= 7 and month <= 9 then
14             set result := '第三季度';
15         -- 条件4
16         when month >= 10 and month <= 12 then
17             set result := '第四季度';
18         -- 上述都不满足
19         else
20             set result := '非法参数';
21         end case ;
22         select concat('您输入的月份为: ',month, ', 所属的季度为: ',result);
23     end;
24 -- 调用存储过程并传入参数
25 call p6(12);

```

信息	结果1	概况	状态
concat('您输入的月份为: ',month, ', 所属的季度为: ',result)			
您输入的月份为: 12, 所属的季度为: 第四季度			

CSDN @观止study

(5) 循环

(5.1) while

while 循环是有条件的循环控制语句。满足条件后，再执行循环体中的SQL语句。具体语法为：

```

WHILE 条件 DO
    -- SQL逻辑...
END WHILE;

```

先判定条件，如果条件为true，则执行逻辑，否则，不执行逻辑。

- 使用示例

```
1
2 -- 创建存储过程
3 create procedure p7(in n int)
4 begin
5 -- 定义局部变量，记录累加之后的值；
6     declare total int default 0;
7 -- 每循环一次，就会对n进行减1，如果n减到0，则退出循环
8     while n > 0 do
9         set total := total + n;
10        set n := n - 1;
11    end while;
12    -- 查看结果
13    select total;
14 end;
15
16 -- 调用存储过程并传入参数
17 call p7(100);
```

信息 结果1 概况 状态

total
5050

CSDN @观止study

(5.2) repeat

与while类似，repeat也是有条件的循环控制语句，当满足until声明的条件的时候，则退出循环。具体语法为：

```
REPEAT
    -- SQL逻辑...
    UNTIL 条件
END REPEAT;
```

先执行一次逻辑，然后判定UNTIL条件是否满足，如果满足，则退出。如果不满足，则继续下一次循环

- 使用示例

```

1
2 -- 创建存储过程
3 create procedure p7(in n int)
4 begin
5 -- 定义局部变量，记录累加之后的值；
6     declare total int default 0;
7 -- 每循环一次，就会对n进行减1，如果n减到0，则退出循环
8 repeat
9     set total := total + n;
10    set n := n - 1;
11    until n <= 0
12 end repeat;
13 -- 查看结果
14 select total;
15 end;
16
17 -- 调用存储过程并传入参数
18 call p7(100);

```

信息 结果1 概况 状态

total
5050

CSDN @观止study

(5.3) loop

LOOP 也可以用来实现循环，如果不在SQL逻辑中增加退出循环的条件，可以用其来实现简单的死循环。LOOP可以配合一下两个语句使用：

- **LEAVE**：配合循环使用，退出循环。
- **ITERATE**：必须用在循环中，作用是跳过当前循环剩下的语句，直接进入下一次循环。

```

[begin_label:] LOOP
    -- SQL逻辑...
    -- LEAVE label; -- 退出指定标记的循环体
    -- ITERATE label; -- 直接进入下一次循环
END LOOP [end_label];

```

begin_label，**end_label**，**label** 指的都是我们所自定义的标记，用来标记loop循环的范围。

- 使用示例：计算从1到n之间的偶数累加的值，n为传入的参数值

```

4  -- 定义局部变量，记录累加之后的值；
5      declare total int default 0;
6  sum:loop
7  -- 每循环一次，就会对n进行-1，如果n减到0
8      if n<=0 then
9          leave sum;  -- 退出循环
10         end if;
11     -- 如果当次累加的数据是奇数
12         if n%2 = 1 then
13             set n := n - 1;
14             iterate sum;  -- 直接进入下一次循环
15         end if;
16         -- 执行至此，说明为偶数，加上
17         set total := total + n;
18         set n := n - 1;
19     end loop sum;
20 select total;
21 end;
22 -- 执行存储过程并传入值
23 call p10(100);

```

息 结果1 概况 状态

total
2550

CSDN @观止study

(5.3) 区别对比

名称	区别
while	先判定条件 ，如果条件为true，则执行逻辑，否则不执行逻辑。
repeat	先执行一次逻辑 ，然后判定UNTIL条件是否满足，如果满足，则退出。如果不满足，则继续下一次循环
loop	可以配合两个语句实现：死循环，退出循环，跳过剩余语句执行下一轮循环

(6) cursor 游标

我们的变量只能存储单个查询结果，而游标（CURSOR）是**用来存储查询结果集**的数据类型，在存储过程和函数中可以使用游标对结果集进行循环的处理。

- 声明游标

```
DECLARE 游标名称 CURSOR FOR 查询语句;
```

- 打开游标

```
OPEN 游标名称;
```

- 获取游标记录

```
FETCH 游标名称 INTO 变量 [,变量 ];
```

- 关闭游标

```
CLOSE 游标名称;
```

使用示例：根据传入的参数uage，来查询用户表tb_user中，所有的用户年龄小于等于uage的用户姓名（name）和专业（profession），并将用户的姓名和专业插入到所创建的一张新表(id,name,profession)中

```
6 declare upro varchar(100);
7 -- 3. 声明游标，存储查询结果集
8 declare u_cursor cursor for select name,profession from user where age <= uage;
9 -- 4. 创建所需表结构
10 create table if not exists user_pro(
11     id int primary key auto_increment,
12     name varchar(100),
13     profession varchar(100)
14 );
15 -- 5. 开启游标
16 open u_cursor;
17 -- 6. 循环获取并插入游标中的记录
18 while true do
19     -- 7. 获取游标中的记录
20     fetch u_cursor into uname,upro;
21     -- 8. 插入数据到新表
22     insert into user_pro values (null,uname,upro);
23 end while;
24 -- 9. 关闭游标
25 close u_cursor;
26 end;
27 -- 调用存储过程
28 call p11(30);
```

插入成功

id	name	profession
1	大乔	舞蹈
2	吕布	软件工程
3	花木兰	软件工程
4	露娜	应用数学

信息 概况 状态

[SQL] call p11(30):

[02000][1329] No data - zero rows fetched, selected, or processed

虽然报错了，但是查看数据库可以看到我们的数据已经成功插入进去了

CSDN @观止study

在我们调用上述存储过程的过程中，由于while循环中并没有退出条件，当游标的数据集获取完毕之后，再次获取数据，就会报错，从而终止了程序的执行。但是此时，user_pro表结构及其数据都已经插入成功了。要想解决这个问题，就需要通过MySQL中提供的 条件处理程序 Handler 来解决

(7) 条件处理程序

条件处理程序（Handler）可以用来定义在流程控制结构执行过程中遇到问题时相应的处理步骤。

- 语法

```
DECLARE handler_action HANDLER FOR condition_value [,condition_value] ... statement;
```

handler_action 的取值：

- CONTINUE：继续执行当前程序
- EXIT：终止执行当前程序

condition_value 的取值：

- SQLSTATE sqlstate_value：状态码，如 02000
- SQLWARNING：所有以01开头的SQLSTATE代码的简写
- NOT FOUND：所有以02开头的SQLSTATE代码的简写
- SQLEXCEPTION：所有没有被SQLWARNING 或 NOT FOUND捕获的SQLSTATE代码的简写

我们来处理完善一下上一个案例所遇到的问题~


```

1  -- 1. 创建存储过程
2  create procedure p11(in uage int)
3  begin
4  -- 2. 声明两个接收数值的变量
5  declare uname varchar(100);
6  declare upro varchar(100);
7  -- 3. 声明游标, 存储查询结果集
8  declare u_cursor cursor for select name,profession from user where age <= uage;
9  -- 当SQL语句执行抛出的状态码为02开头时, 将关闭游标u_cursor, 并退出
10 declare exit handler for not found close u_cursor;
11 -- 4. 创建所需表结构
12 create table if not exists user_pro(
13     id int primary key auto_increment,
14     name varchar(100),
15     profession varchar(100)
16 );
17 -- 5. 开启游标
18 open u_cursor;
19 -- 6. 循环获取并插入游标中的记录
20 while true do
21     -- 7. 获取游标中的记录
22     fetch u_cursor into uname,upro;
23     -- 8. 插入数据到新表
24     insert into user_pro values (null,uname,upro);
25 end while;
26 -- 9. 关闭游标
27 close u_cursor;
28 end;
29 -- 调用存储过程
30 call p11(30);

```

异常处理, 通过报错语句我们可以得知报错码为 02000

成功插入

id	name	profession
1	吕布	软件工程
2	花木兰	软件工程
3	大乔	舞蹈
4	露娜	应用数学

信息 概况 状态

[SQL] call p11(30);
 受影响的行: 0
 时间: 0.020s

没有报错

CSDN @观止study

• 多种写法示例:

```

-- 当SQL语句执行抛出的状态码为02000开头时, 将关闭游标u_cursor, 并退出
declare exit handler for SQLSTATE '02000' close u_cursor;

-- 当SQL语句执行抛出的状态码为02开头时, 将关闭游标u_cursor, 并退出
declare exit handler for not found close u_cursor;

-- 当SQL语句执行抛出的异常没有其他处理程序处理时, 将关闭游标u_cursor, 并退出
declare exit handler for SQLEXCEPTION close u_cursor;

```

常见错误状态码可参考[官方文档](#)。

四. 存储函数

存储函数是有返回值的存储过程, 存储函数的参数只能是 in 类型的。具体语法如下:

```

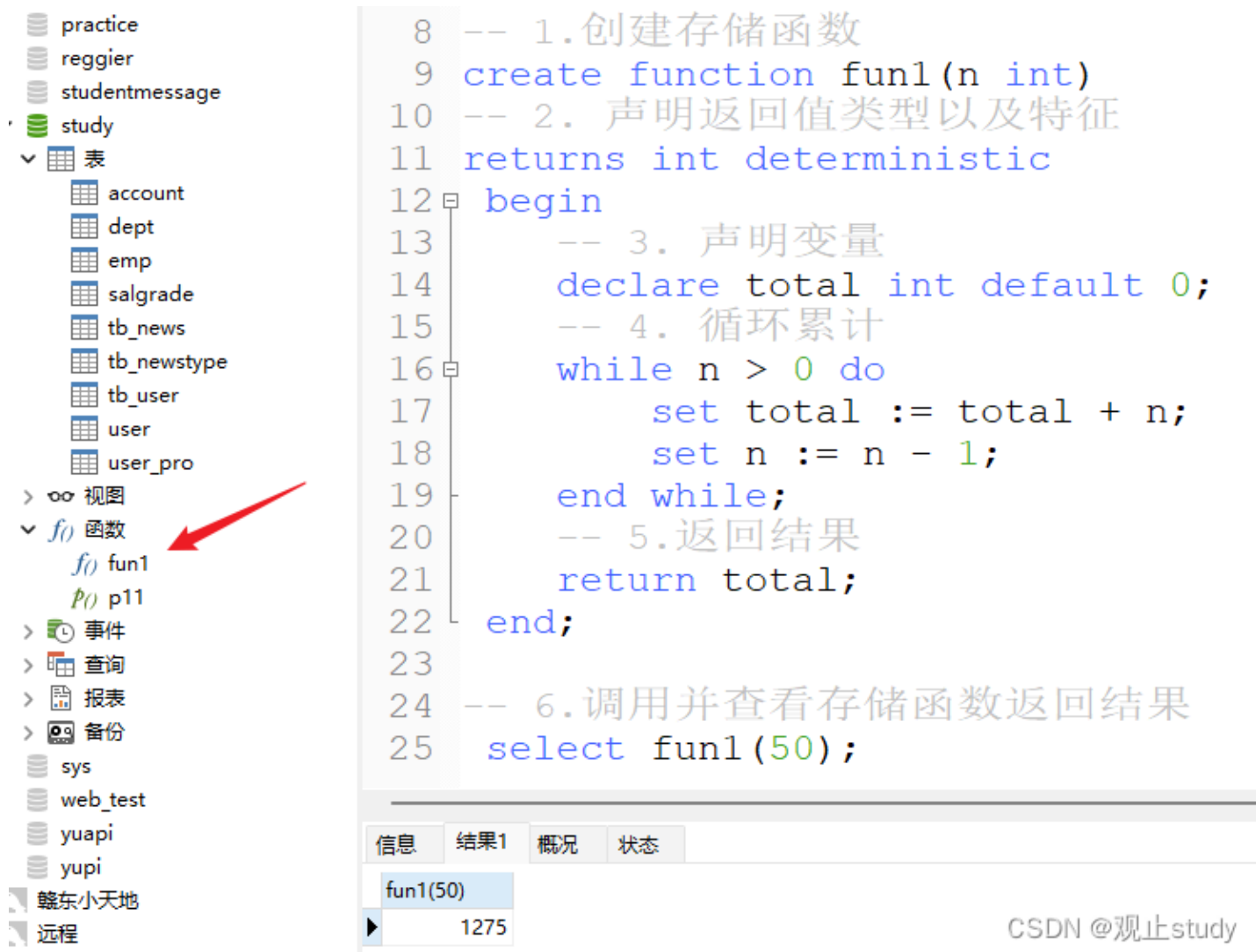
CREATE FUNCTION 存储函数名称 ([ 参数列表 ])
RETURNS type [characteristic ...]
BEGIN
    -- SQL语句
    RETURN ...;
END ;

```

characteristic参数说明：

- **DETERMINISTIC**：相同类型的输入参数总是产生相同类型的结果
- **NO SQL**：不包含 SQL 语句。
- **READS SQL DATA**：包含读取数据的语句，但不包含写入数据的语句。

使用示例：计算从1累加到n的值，n为传入的参数值



The screenshot shows a MySQL IDE interface. On the left is a sidebar with a tree view of the database structure. The 'study' database is selected, and the '函数' (Functions) folder is expanded, showing a function named 'fun1'. A red arrow points to 'fun1'. The main area displays the SQL code for creating and calling the function. The code is as follows:

```
8 -- 1.创建存储函数
9 create function fun1(n int)
10 -- 2. 声明返回值类型以及特征
11 returns int deterministic
12 begin
13     -- 3. 声明变量
14     declare total int default 0;
15     -- 4. 循环累计
16     while n > 0 do
17         set total := total + n;
18         set n := n - 1;
19     end while;
20     -- 5.返回结果
21     return total;
22 end;
23
24 -- 6.调用并查看存储函数返回结果
25 select fun1(50);
```

Below the code editor, there is a results pane with tabs for '信息' (Information), '结果1' (Result 1), '概况' (Overview), and '状态' (Status). The '结果1' tab is active, showing the output of the function call: 'fun1(50)' with the value '1275'.

CSDN @观止study

注意：在mysql8.0版本中 **binlog** 默认是开启的，一旦开启了，mysql就要求在定义存储过程时，需要指定 characteristic 特性，否则就会报如下错误：

[HY000][1418] This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

五.全文概览

存储过程是事先经过编译并存储在数据库中的一段 SQL 语句的集合，相当于数据库 SQL 语言层面的代码封装与重用。

一.概述

```
CREATE PROCEDURE 存储过程名称 ([ 参数列表 ])
BEGIN
-- SQL语句
END;
```

1.在客户端创建

(1) 创建

2.在命令行创建

语法与上述一致，但需要先通过关键字
delimiter重新指定SQL语句的结束符

(2) 调用

CALL 名称 ([参数列表]);

(3) 查看

方式一

```
SELECT * FROM INFORMATION_SCHEMA.
ROUTINES WHERE ROUTINE_SCHEMA = '数
据库名称';
```

方式二

```
SHOW CREATE PROCEDURE 存储过程名称;
```

(4) 删除

```
DROP PROCEDURE [ IF EXISTS ] 存储过程名
称;
```

二.基础语法

(1) 作用域

1.全局变量

设置后针对于所有的会话生效

2.会话变量

只对当前会话生效，在另外一个会话窗口就不生效了

(2) 查看变量

1.查看所有系统变量

```
SHOW [ SESSION | GLOBAL ] VARIABLES;
```

2.通过LIKE模糊匹配方式查找变量

```
SHOW [ SESSION | GLOBAL ] VARIABLES
LIKE '.....';
```

3.查看指定变量的值

```
SELECT @@[SESSION | GLOBAL] 系统变量名;
```

(3) 设置变量

方式一

```
SET [ SESSION | GLOBAL ] 系统变量名 = 值;
```

方式二

```
SET @@[SESSION | GLOBAL]系统变量名 =
值;
```

(1) 变量

1.系统变量

2.用户定义变量

(1) 赋值 (声明)

方式一

```
SET @var_name = expr [, @var_name =
expr] ...;
```

方式二

```
SET @var_name := expr [, @var_name :=
expr] ...;
```

方式三

```
SELECT @var_name := expr [, @var_
name := expr] ...
```

方式四

```
SELECT 字段名 INTO @var_name FROM 表名;
```

(2) 查看变量

```
SELECT @变量名;
```

3.局部变量

(1) 声明

```
create procedure 存储过程名称 ([ 参数列表 ])
begin
declare 变量名 变量类型 [DEFAULT ...];
-- ....
-- SQL语句
end;
```

(2) 赋值

方式一

```
SET 变量名 = 值;
```

方式二

```
SET 变量名 := 值;
```

方式三

```
SELECT 字段名 INTO 变量名 FROM 表名 ...;
```

(2) if 判断

1.完整版

```
IF 条件1 THEN
.....
ELSEIF 条件2 THEN  -- 可选
.....
ELSE  -- 可选
.....
END IF;
```

2.简洁版

```
IF 条件1 THEN
.....
END IF;
```

(3) 参数

1.类型

IN

输入参数，调用时需要传入值

OUT

输出参数，该参数可以作为返回值

INOUT

既可以作为输入参数，也可以作为输出参数

2.使用

```
CREATE PROCEDURE 存储过程名称 ([ [IN |
OUT | INOUT] 参数名 参数类型 ])
BEGIN
-- SQL语句
END;
```

(4) case 流程控制

1.数值控制

```
CASE case_value
WHEN when_value1 THEN statement_list1
[ WHEN when_value2 THEN statement_
list2] ...
[ ELSE statement_list]
END CASE;
```

2.条件控制

```
CASE
WHEN search_condition1 THEN
statement_list1
[ WHEN search_condition2 THEN
statement_list2] ...
[ ELSE statement_list]
END CASE;
```

1.while

特点

先判定条件，如果条件为true，则执行逻辑，否则，不执行逻辑。

语法

```
WHILE 条件 DO
-- SQL逻辑...
END WHILE;
```

特点

先执行一次逻辑，然后判定UNTIL条件是否满足，如果满足，则退出。如果不满足，则继续下一次循环

十一.存储过程

三.进阶使用

五. 参数函数

(1) 用途 有返回值的存储过程，存储函数的参数只能是in类型的

(2) 语法
`CREATE FUNCTION 存储函数名称 ([参数列表])
RETURNS type [characteristic ...]
BEGIN
-- SQL语句
RETURN ...;
END;`

(3) characteristic 参数说明

DETERMINISTIC 相同类型的输入参数总是产生相同类型的结果

NO SQL 不包含 SQL 语句

READS SQL DATA 包含读取数据的语句，但不包含写入数据的语句。

(5) 循环

2.repeat

语法

```
REPEAT;  
-- SQL逻辑...  
UNTIL 条件  
END REPEAT;
```

特点 可以配合两个语句实现：死循环，退出循环，跳过剩余语句执行下一轮循环

3.loop

辅助语句

LEAVE 配合循环使用，退出循环

ITERATE 必须用在循环中，作用是跳过当前循环剩下的语句，直接进入下一次循环

语法

```
[begin_label:] LOOP  
-- SQL逻辑...  
-- LEAVE label; -- 退出指定标记的循环体  
-- ITERATE label; -- 直接进入下一次循环  
END LOOP [end_label];
```

(6) cursor 游标

1.用途 用来存储查询结果集

2.声明游标 `DECLARE 游标名称 CURSOR FOR 查询语句;`

3.打开游标 `OPEN 游标名称;`

4.获取游标记录 `FETCH 游标名称 INTO 变量 [变量];`

5.关闭游标 `CLOSE 游标名称;`

(7) 条件处理程序

1.用途 可以用来定义在流程控制结构执行过程中遇到问题时相应的处理步骤

2.语法

```
DECLARE handler_action HANDLER FOR  
condition_value [,condition_value] ...  
statement;
```

3.相关参数取值说明

handler_action

CONTINUE 继续执行当前程序

EXIT 终止执行当前程序

condition_value

SQLSTATE sqlstate_value 状态码，如 02000

SQLWARNING 所有以01开头的SQLSTATE代码的简写

NOT FOUND 所有以02开头的SQLSTATE代码的简写

SQLEXCEPTION 所有没有SQLWARNING 或 NOT FOUND捕获的SQLSTATE代码的简写