19 听过存储过程，但是你会用吗？

更新时间：2020-04-15 09:42:37

> 生活永远不像我们想像的那样好，但也不会像我们想像的那样糟。——莫泊桑

存储过程是 MySQL 提供的高级特性，是在 5.0 版本之后加入的，所以，我们现在使用的 MySQL 都基本支持存储过程。存储过程就像是一门程序设计语言，它同样包含了数据类型、流程控制、输入输出等等特性。不过，对于很多人来说，存储过程会比较陌生，对它的概念仅仅停留在“听说”上。于是，这一节里，我不仅要讲解存储过程是什么，还要让你理解怎么去使用它。

1. 存储过程概述

学习任何知识都不应该着急，否则，一定是“欲速则不达”的。对于存储过程这种高级特性，我们平时的业务系统使用的较少，甚至是不用，就一定不要着急去操作它。先跟我一起把它的概念搞清楚，知道它是什么、能做什么之后，再做其他打算。

1.1 什么是存储过程

存储过程是一组用于完成特定功能的 SQL 语句集（注意，这句话是重点），经过编译之后存储在数据库中。存储过程是可编程的函数，可以简单理解为 SQL 的组合，并加上了逻辑控制。在调用的时候，需要指定存储过程的名称以及参数（如果存储过程带有参数）。

存储过程的调用方（使用方）可以是触发器，也可以是 Java、Python、PHP 等等应用程序，当然还包括其他的存储过程。但是，需要注意，存储过程不能调用自己，即 MySQL 并不支持递归的存储过程。

1.2 存储过程的优缺点

任何技术都有自己的优势，也有自己的劣势，存储过程当然也不例外。下面，我会去总结存储过程的优缺点，而你也要去理解它们，思考下什么场景适合使用存储过程。

存储过程的优点

- 编译之后存储在数据库中，执行速度快，减少网络中代码（SQL 语句）的传输
- 可以重复使用，但是不需要重复编写代码，即有可重用性
- 透明性，使用存储过程的用户不需要关心它的实现过程
- 安全性强，可以只单独授予执行存储过程的权限，而不提供任何代码中涉及的库、表等权限

存储过程的缺点

- MySQL 并不提供调试存储过程的功能，出现问题，难以定位
- 对于复杂逻辑的存储过程，开发和维护的难度太大
- 可移植性差

存储过程的思想和意图并不难理解，之所以对它不熟悉，甚至是望而生畏，主要原因就是因为没用过，也没见别人用过。那么，接下来，跟着我一起操作下存储过程吧。

2. 存储过程的相关操作

编写存储过程讲究的还是挺多的：它的语法、是否有参数、参数又会有哪些特性等等。下面，我们就重点来把编写存储过程的知识点搞清楚。而对于调用、删除和查看存储过程就非常简单了。

2.1 存储过程的语法

这里，我先给出 MySQL 官方定义的存储过程语法：

```
CREATE PROCEDURE p_name()
  BEGIN
  [statement_list]
  END
```

但是，大多数时候，我们写存储过程时都会去修改数据库的标准分隔符（结尾的分号）。所以，我想以一个简单的存储过程来说明它的语法。如下所示（它是可以执行的）：

```
DELIMITER $$
 CREATE PROCEDURE one()
   BEGIN
   SELECT *  FROM worker;
   END $$
DELIMITER ;
```

这个语法确实是不好理解，下面，我来对它解读下（复杂的存储过程也只是逻辑复杂，语法肯定都是一样的，所以，重点理解它的语法，而不在于它的实现）：

- 开头的 “DELIMITER KaTeX parse error: Expected 'EOF', got '"' at position 1: "_是与存储过程无关的，它其实是…”。之所以需要这样做，是想要将存储过程作为整体传递给服务器，而不是让 MySQL 解释每条语句
- “CREATE PROCEDURE” 标识创建一个新的存储过程，后面的 “one” 是存储过程的名称。最后，还需要跟一对括号，这有点类似于函数

- “BEGIN” 和 “END” 之间的部分称为存储过程主体（注意 END 后面需要加上分隔符），也就是业务逻辑。这里，我简单的查询 worker 表中的数据
- 最后一句 “DELIMITER ;” 也是与存储过程无关的，它的目的是将分隔符更改回分号

就像我们日常写代码一样，不论这段代码多复杂，都不会超出语言自身语法的限制。复杂的部分只是业务逻辑，无非就是 if、else、for 的组合。所以，先去理解语法，再去逐步拆解业务逻辑。

2.2 无参存储过程

存储过程类似于函数，也是分为有参和无参的，我们先去看一看无参的存储过程，它也相对简单许多。在编写示例之前，我们先要准备一张 worker 表，表中存储如下数据（这里没有很多限制，可以是任意的表，任意的数据）：

```
mysql> SELECT * FROM worker;
+----+------+--------+--------+---------+
| id | type | name   | salary | version |
+----+------+--------+--------+---------+
|  1 | A    | tom    |   1800 |       0 |
|  2 | B    | jack   |   2100 |       0 |
|  3 | C    | pony   |   NULL |       0 |
|  4 | B    | tony   |   3600 |       0 |
|  5 | B    | marry  |   1900 |       0 |
|  6 | C    | tack   |   1200 |       0 |
|  7 | A    | tick   |   NULL |       0 |
|  8 | B    | clock  |   2000 |       0 |
|  9 | C    | noah   |   1500 |       0 |
| 10 | C    | jarvis |   1800 |       0 |
+----+------+--------+--------+---------+
10 rows in set (0.00 sec)
```

如果我想查出所有的 worker 中，最高和最低 salary 的记录，但是又不想总是写 SQL 语句，就可以写成无参的存储过程。如下所示：

```
DELIMITER $$
CREATE PROCEDURE max_min_salary_from_worker()
BEGIN
   SELECT MAX(salary), MIN(salary) FROM worker;
END $$
DELIMITER ;
```

可以把上面的 SQL 语句放到 MySQL 客户端中执行，也就在当前库中创建了 max_min_salary_from_worker 存储过程。既然已经创建了，我们就来调用下看看结果是否符合预期吧。

```
-- 调用存储过程使用 CALL 命令，后面跟着名称和括号（和高级语言中调用函数、方法类似）
mysql> CALL max_min_salary_from_worker();
+-------------+-------------+
| MAX(salary) | MIN(salary) |
+-------------+-------------+
|        3600 |        1200 |
+-------------+-------------+
1 row in set (0.02 sec)
```

2.3 有参存储过程

存储过程根据需要可能会有输入、输出以及输入输出参数，这也是很常见的需求。MySQL 为了支持此项功能，提供了三种类型的参数，

- IN：传递给存储过程的参数，在存储过程中修改参数值不能返回，即输入

- OUT：存储过程传出的参数，可在存储过程内部被改变，并可返回，即输出
- INOUT：存储过程的传入和传出参数，可被改变和返回，即输入输出

接下来，我将给出几个示例存储过程，依次来看一看这些类型的参数怎么应用。首先，看一看 IN，仅带入参的存储过程（执行过程带有注释信息）：

```
-- 修改分隔符为 $$
mysql> DELIMITER $$

-- 创建带有入参的存储过程，注意，在存储过程中对入参进行了修改
mysql>  CREATE PROCEDURE query_worker_by_salary(IN sal INT)
    ->  BEGIN
    ->  SELECT * FROM worker WHERE salary > sal;
    ->  SET @sal = 5000;
    ->  END $$
Query OK, 0 rows affected (0.02 sec)

-- 将分隔符修改回分号
mysql> DELIMITER ;

-- 设置变量 sal 的值为 2000
mysql> SET @sal = 2000;
Query OK, 0 rows affected (0.01 sec)

-- 调用存储过程并传入参数 sal
mysql> CALL query_worker_by_salary(@sal);
+----+------+------+--------+----------+
| id | type | name | salary | version |
+----+------+------+--------+----------+
|  2 | B    | jack |   2100 |       0 |
|  4 | B    | tony |   3600 |       0 |
+----+------+------+--------+----------+
2 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

-- 再次查看变量 sal，发现在存储过程中修改并未生效
mysql> SELECT @sal;
+------+
| @sal |
+------+
| 2000 |
+------+
1 row in set (0.00 sec)
```

如果想要在存储过程中修改某个变量并带回来，就需要使用到 OUT 类型的参数。同样，我给出一个示例存储过程的执行流程，并带有相应的解释说明。

```
-- 修改分隔符为 $$
mysql> DELIMITER $$

-- 创建带有出参的存储过程，将 worker 记录最大的 id 传递给参数 max_id
mysql>  CREATE PROCEDURE query_worker_max_id(OUT max_id INT)
    ->  BEGIN
    ->  SELECT MAX(id) INTO max_id FROM worker;
    ->  END $$
Query OK, 0 rows affected (0.02 sec)

-- 将分隔符修改回分号
mysql> DELIMITER ;

-- 调用存储过程并传递参数 max_id
mysql> CALL query_worker_max_id(@max_id);
Query OK, 1 row affected (0.00 sec)

-- 查看 max_id，已经从存储过程中获取了最大记录的 id
mysql> SELECT @max_id;
+---------+
| @max_id |
+---------+
|      10 |
+---------+
1 row in set (0.00 sec)
```

存储过程当然可以有多个参数，多个参数也可以是不同类型，它们需要使用“,”分隔开。那么，想要查看下 worker 表中 salary 大于 x(例如2000) 的记录数有多少，可以这样去编写和执行存储过程：

```
-- 修改分隔符为 $$
mysql> DELIMITER $$

-- 创建存储过程，包含两个参数：入参 sal，出参 cnt
mysql>  CREATE PROCEDURE query_worker_count_by_salary(IN sal INT, OUT cnt INT)
    ->  BEGIN
    ->  SELECT COUNT(id) INTO cnt FROM worker WHERE salary > sal;
    ->  END $$
Query OK, 0 rows affected (0.02 sec)

-- 将分隔符修改回分号
mysql> DELIMITER ;

-- 调用存储过程，传递入参为 2000，出参为 cnt
mysql> CALL query_worker_count_by_salary(2000, @cnt);
Query OK, 1 row affected (0.01 sec)

-- 查看 cnt，确定 salary 大于 2000 的记录数有多少
mysql> SELECT @cnt;
+------+
| @cnt |
+------+
|    2 |
+------+
1 row in set (0.00 sec)
```

最后，对于 query_worker_count_by_salary 我们也可以使用 INOUT 类型的参数实现。由于 INOUT 类型兼具入参和出参的功能，所以，我们只需要定义一个变量即可。重新实现的存储过程和执行流程如下所示：

```
-- 修改分隔符为 $$
mysql> DELIMITER $$

-- 创建存储过程，并指定 INOUT 类型的 sal_cnt 参数（注意参数的使用与赋值过程）
mysql>  CREATE PROCEDURE query_worker_count_by_salary_2(INOUT sal_cnt INT)
    ->  BEGIN
    ->  SELECT COUNT(id) INTO sal_cnt FROM worker WHERE salary > sal_cnt;
    ->  END $$
Query OK, 0 rows affected (0.02 sec)

-- 将分隔符修改回分号
mysql> DELIMITER ;

-- 设置变量 sal_cnt 的值为 2000
mysql> SET @sal_cnt = 2000;
Query OK, 0 rows affected (0.00 sec)

-- 调用存储过程，并传递参数 sal_cnt
mysql> CALL query_worker_count_by_salary_2(@sal_cnt);
Query OK, 1 row affected (0.00 sec)

-- 查看 sal_cnt，确定 salary 大于 2000 的记录数有多少
mysql> SELECT @sal_cnt;
+----------+
| @sal_cnt |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

经过以上实例的学习，你应该知道了怎样定义与使用存储过程。无参的存储过程是比较简单的，相对来说，它的使用频率也不会很高。有参的存储过程主要是理解三种类型参数的思想，同时也正是由于它支持参数的传递，也更加的灵活，可用性也就更高。

2.4 删除存储过程

存储过程是可以修改的，但是只能改变存储过程的特征（注释信息和权限），不能修改存储过程的参数和主体。所以，修改存储过程的意义是不大的。更合理的做法肯定是删除掉原来的存储过程，再去重新创建新的。

关于删除存储过程，当然也是正常的需求。但是，需要特别注意：不能在一个存储过程中删除另一个存储过程，只能调用另一个存储过程。删除的语法和执行过程（带有注释信息）如下所示：

```
-- 删除存储过程的语法，如果是在当前的数据库中，db_name 可以不写
DROP PROCEDURE [IF EXISTS] db_name.p_name;

-- 删除存储过程 query_worker_count_by_salary_2
mysql> DROP PROCEDURE IF EXISTS `imooc_mysql`.`query_worker_count_by_salary_2`;
Query OK, 0 rows affected (0.04 sec)

-- 验证存储过程已经被删除
mysql> CALL query_worker_count_by_salary_2(@sal_cnt);
ERROR 1305 (42000): PROCEDURE imooc_mysql.query_worker_count_by_salary_2 does not exist
```

2.5 查看存储过程

存储过程的定义信息保存在 information_schema 库的 ROUTINES 表和 mysql 库的 proc 表中。我们先来看一看这两张表中保存了哪些信息（SQL 语句过长，做了格式化处理）：

```
-- 可以通过 information_schema.ROUTINES 表查看存储过程的名称和定义信息
mysql> SELECT
    ->   ROUTINE_NAME,
    ->   ROUTINE_TYPE,
    ->   ROUTINE_DEFINITION
    -> FROM
    ->   information_schema.ROUTINES
    -> WHERE
    ->   ROUTINE_SCHEMA = 'imooc_mysql'
    ->   AND ROUTINE_NAME = 'max_min_salary_from_worker'\G
*************************** 1. row ***************************
     ROUTINE_NAME: max_min_salary_from_worker
     ROUTINE_TYPE: PROCEDURE
ROUTINE_DEFINITION: BEGIN
  SELECT MAX(salary), MIN(salary) FROM worker;
END
1 row in set (0.00 sec)

-- 可以通过 mysql.proc 表查看存储过程的名称、主体、参数列表、创建者等等信息
mysql> SELECT
    ->   name,
    ->   body,
    ->   param_list,
    ->   definer
    -> FROM
    ->   mysql.proc
    -> WHERE
    ->   db = 'imooc_mysql'
    ->   AND type = 'PROCEDURE' LIMIT 1\G
*************************** 1. row ***************************
      name: max_min_salary_from_worker
      body: BEGIN
  SELECT MAX(salary), MIN(salary) FROM worker;
END
param_list:
   definer: root@localhost
1 row in set (0.00 sec)
```

查询系统表获取到存储过程信息当然是可以的，但是，SQL 语句写起来比较长，而且系统表名也不便于记忆。所以，我们通常会使用 `SHOW CREATE PROCEDURE` 命令查看存储过程的信息。如下所示：

```
mysql> SHOW CREATE PROCEDURE `imooc_mysql`.`query_worker_count_by_salary`\G
*************************** 1. row ***************************
           Procedure: query_worker_count_by_salary
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
    Create Procedure: CREATE DEFINER=`root`@`localhost` PROCEDURE `query_worker_count_by_salary`(IN sal INT, OUT cnt INT)
BEGIN
  SELECT COUNT(id) INTO cnt FROM worker WHERE salary > sal;
  END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

如果仅仅是想要查看某个数据库中定义了哪些存储过程，而不关心它们实现的主体，有更简单的办法：

```
-- 通过 SHOW PROCEDURE STATUS 命令可以查看到某个数据库下面定义的存储过程的基本信息
mysql> SHOW PROCEDURE STATUS WHERE db = 'imooc_mysql';
+-------------+-----------------------------+-----------+----------------+---------------------+---------------------+---------------+---------+----------------------+----------------------+--------------------+
| Db          | Name                        | Type      | Definer        | Modified            | Created             | Security_type | Comment | character_set_client | collation_connection | Database Collation |
+-------------+-----------------------------+-----------+----------------+---------------------+---------------------+---------------+---------+----------------------+----------------------+--------------------+
| imooc_mysql | max_min_salary_from_worker  | PROCEDURE | root@localhost | 2019-12-12 10:39:15 | 2019-12-12 10:39:15 | DEFINER       |         | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
| imooc_mysql | one                         | PROCEDURE | root@localhost | 2019-12-11 23:52:56 | 2019-12-11 23:52:56 | DEFINER       |         | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
| imooc_mysql | query_worker_by_salary      | PROCEDURE | root@localhost | 2019-12-12 11:25:01 | 2019-12-12 11:25:01 | DEFINER       |         | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
| imooc_mysql | query_worker_count_by_salary | PROCEDURE | root@localhost | 2019-12-12 13:09:50 | 2019-12-12 13:09:50 | DEFINER       |         | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
| imooc_mysql | query_worker_max_id         | PROCEDURE | root@localhost | 2019-12-12 11:45:55 | 2019-12-12 11:45:55 | DEFINER       |         | utf8                 | utf8_general_ci      | latin1_swedish_ci  |
+-------------+-----------------------------+-----------+----------------+---------------------+---------------------+---------------+---------+----------------------+----------------------+--------------------+
5 rows in set (0.01 sec)
```

3. 存储过程与游标

我们在日常的工作中很少听说“游标”这个概念，因为它操作起来相对比较复杂，大多数人更愿意使用代码去解决问题。也不像多数 DBMS，MySQL 的游标只能应用于存储过程和函数。所以，游标的出场率就更低了。不过，想要编写逻辑复杂的存储过程，几乎都离不开游标的帮忙，几经思考，我还是想去讲解下游标的概念和怎么使用游标。

3.1 初识游标

既然想要知道游标能做什么，就需要先搞明白游标是什么，先来看一看关于游标的定义：

> 游标（Cursor）是一个存储在服务器上的数据库查询，但是，它并不是一条 SELECT 语句，而是被该语句检索出来的结果集。游标可以看做是指向查询结果集的指针，通过游标，我们可以一次一行的处理结果集的数据。

也就是说，游标是交互式的应用，用于滚动查询数据，并对数据进行浏览或更改操作。那么，关于它的应用也就很好理解了：当结果集中存在多行数据时，如果想以行为单位进行处理，就必须要使用到游标。

3.2 游标的使用过程

游标的使用或者说处理过程一共有四步，当然，你也可以认为这就是游标的语法。下面，我对这四步过程进行解读（需要好好理解，搞清楚了思想和语法，余下的就只是业务逻辑了）：

- 声明游标

声明游标需要两个元素：游标的名称和 SELECT 查询的结果集。其语法如下：

```
-- 需要注意，游标声明必须出现在变量和条件的后面，且一个存储过程可以声明多个游标
DECLARE cursor_name CURSOR FOR select_statement;
```

- 打开游标

打开游标用以将声明时 SELECT 的查询实际检索出来（可见，游标声明的查询时 Lazy 模式的），只需要提供游标名即可。其语法如下：

```
OPEN cursor_name;
```

- 检索游标

检索游标是从游标中取出一行（SELECT 查询的数据记录），并把该行的各个列值保存到各个变量（变量是需要在存储过程中自行定义的）中。检索的特性是一次只取一行，取完之后，自动移动指针到下一行。但是，如果没有拿到行记录，则会抛出异常，对应的 SQLSTATE 代码值为 “02000”。此时，需要在存储过程中声明异常处理程序（声明 NOT FOUND 错误也可以）。其语法如下：

```
FETCH cursor_name INTO var_name [, var_name] ...
```

- 关闭游标

游标会占用数据库的内存和资源，使用完之后需要关闭它。在一个游标关闭后，如果没有重新打开，则不能使用它。但是，声明过的游标不需要再次声明，用 OPEN 语句打开它就可以了。其语法如下：

```
CLOSE cursor_name;
```

3.3 游标在存储过程中的应用

知道了什么是游标以及游标的使用过程，我们就可以尝试着去应用下游标了。我有这样一个需求：想要把 worker 表中 salary 大于等于 2000 的记录存储到 high_income_worker 表（一张新表）中，且只存储 name 和 salary。那么，使用存储过程结合游标可以这样实现（带有详细的注释）：

```
-- 修改分隔符为 $$
DELIMITER $$

-- 创建存储过程，需要传递参数
CREATE PROCEDURE find_condition_salary_worker(IN sal INT)
BEGIN
 -- 声明循环控制变量
 DECLARE v_done BOOLEAN DEFAULT 0;
 -- 声明 v_name，v_salary 变量用于存储 worker 表的两列数据
 DECLARE v_name varchar(64);
 DECLARE v_salary INT;

 -- 声明游标（一定要在变量声明之后），并指定查询结果集
 DECLARE w_cursor CURSOR FOR SELECT name, salary FROM worker WHERE salary >= sal;
 -- 出现 02000 错误时把循环控制变量（v_done）的值设置为 1
 DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET v_done = 1;

 -- 当 high_income_worker 表不存在时创建，用于存储游标结果集
 CREATE TABLE IF NOT EXISTS high_income_worker (name varchar(64), salary int);

 -- 打开游标
 OPEN w_cursor;

 -- 开始循环
 REPEAT
  -- 检索游标，从游标中取出一行，并把列数据赋值给变量
  FETCH w_cursor into v_name, v_salary;
  -- 判断是否遇到终止错误，如果没有
  IF NOT v_done THEN
   -- 将游标结果数据插入到 high_income_worker 表中
   INSERT INTO high_income_worker(name, salary) VALUES(v_name, v_salary);
  -- 终止 IF
  END IF;
 -- 直到遇到终止错误停止循环
 UNTIL v_done END REPEAT;

 -- 关闭游标
 CLOSE w_cursor;
END $$

-- 将分隔符修改回分号
DELIMITER ;
```

在 MySQL 客户端中执行以上语句创建存储过程之后，可以调用并验证结果是否符合预期。如下所示：

```
-- 调用存储过程，并传递参数 2000
mysql> CALL find_condition_salary_worker(2000);
Query OK, 0 rows affected (0.11 sec)

-- 直接查询 high_income_worker 表，验证表存在且数据符合预期
mysql> SELECT * FROM high_income_worker;
+-------+--------+
| name  | salary |
+-------+--------+
| clock |   2000 |
| jack  |   2100 |
| tony  |   3600 |
+-------+--------+
3 rows in set (0.00 sec)
```

4. 总结

存储过程在理解和使用上有一定的难度，但是它却是“一劳永逸”的。我们可以尝试着将常用且复杂的 SQL 查询编写为存储过程，简洁操作的同时也增强了安全性。但是，存储过程也存在着很严重的缺陷，MySQL 并不支持对它的调试，以至于一旦出现错误，排查起来非常困难。所以，在对存储过程的使用上，需要权衡利弊。

5. 问题

你使用过存储过程吗 ？ 是怎么使用的呢 ？ 如果没有，你能说出存储过程的适用场景吗 ？

为了保证存储过程的安全性，需要授予用户可以执行的权限，语法如下所示：

```
GRANT EXECUTE ON PROCEDURE <存储过程名> TO <user>
```

你能根据这个语法，对你创建的存储过程授权给其他用户吗 ？

6. 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：CREATE PROCEDURE and CREATE FUNCTION Statements

MySQL 官方文档：Stored Procedures and Functions

MySQL 官方文档：DROP PROCEDURE and DROP FUNCTION Statements

MySQL 官方文档：SHOW PROCEDURE CODE Statement

MySQL 官方文档：SHOW PROCEDURE STATUS Statement

}

18 外键是一个非常特殊的存在

20 数据汇总优化查询方案设计