 我们知道，在MySQL的储存过程中，一般的输入参数为各种数据类型，如数值型，字符型等。那么，能不能让输入参数等于表名呢？答案是肯定的，那就是使用MySQL提供的PREPARE语法。  
  关于PREPARE的具体使用方法，可以参考网址：<https://dev.mysql.com/doc/refman/5.6/en/sql-syntax-prepared-statements.html> .本文将会给出一个在MySQL的储存过程中使用表名为变量的例子。  
  打开Navicat For MySQL，在新建查询中，创建示例的储存过程sp1，代码如下：

**DROP** **PROCEDURE** **IF** **EXISTS** sp1;

DELIMITER ;;

**CREATE** **PROCEDURE** sp1(**IN** temp **varchar**(30))

**BEGIN**

#Routine body goes here...

**SET** @sqlStr:=CONCAT("select \* from ",temp);

PREPARE stmt from @sqlStr;

EXECUTE stmt;

DEALLOCATE PREPARE stmt;

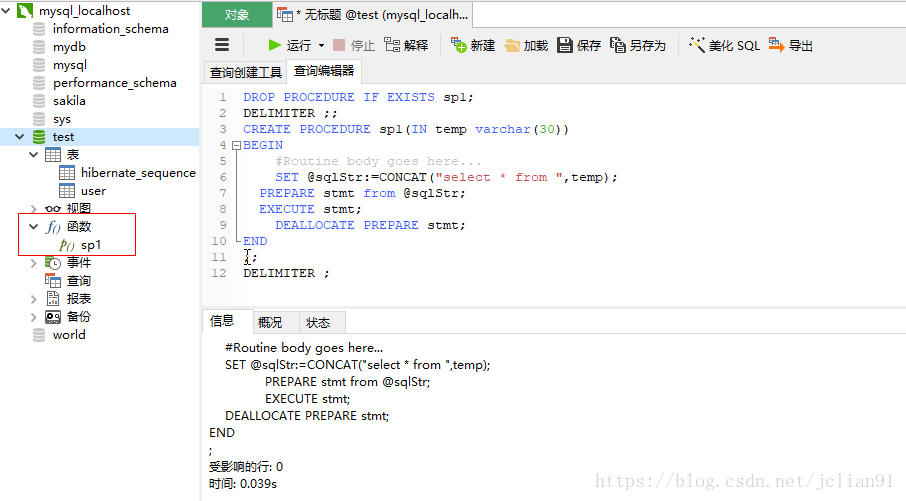
**END**

;;

DELIMITER ;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

该储存过程使用temp为表名，将查询语句“select \* from ” , temp拼接为字符串，赋值给sqlStr变量，再用PREPARE将该字符串转化为MySQL脚本中的Prepared Statements stmt， 然后执行stmt即可获得输出结果。最后通过DEALLOCATE释放该Prepared Statements.  
  运行该查询语句，可发现新建了储存过程sp1，如下：



  新建查询，调用该储存过程：

**call** sp1("user");

* 1

运行结果如下：



  运行成功！

今天写存储过程时，遇到要将表名最为参数的问题，如果不涉及到游标的话，使用prepare可以解决问题，但是，动态表名要运用在游标中的话，则prepare就得靠边站了。

集众人之智慧，最后，使用临时表解决了问题。

**如何在**[MySQL](http://lib.csdn.net/base/mysql)**的存储过程中实现把过程参数用在游标定义的SELECT命令里面作为表名引用**

首先，我们来把场景描绘一下，比如下面的例子（当然是无法正确运行的）：

2. CREATE PROCEDURE `proc`(SourceDBName CHAR(50), SourceTableName CHAR(50),
3. TargetDBName CHAR(50), TargetTemplateTableName CHAR(50))
4. BEGIN
5. DECLARE done INT DEFAULT 0;
6. DECLARE FieldValue CHAR(50);
7. DECLARE CursorSegment CURSOR FOR SELECT ... FROM SourceDBName.SourceTableName;
8. DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
10. OPEN CursorSegment;
11. REPEAT
12. FETCH CursorSegment INTO FieldValue;
13. IF NOT done THEN
14. ...
15. END IF;
16. UNTIL done END REPEAT;
17. CLOSE CursorSegment;
18. END$$

上面的例子试图通过存储过程的参数传递，向存储过程内部的游标定义传递要SELECT的[数据库](http://lib.csdn.net/base/mysql)名称和表名称。但是，这个存储过程在运行时MySQL会提示“SourceDBName.SourceTableName”不存在。也就是说MySQL不会把SourceDBName和SourceTableName两个标识符作为局部变量去解析，而是直接作为表引用。

要解决这个问题，唯一的方法就是把上面这个存储过程分为3个存储过程。对，3个。所以说这是一个比较复杂的解决办法。

第一个存储过程，扮演的是数据收集器的角色。它接收参数传递过来的数据库名和表名，然后把数据SELECT到一个临时表中。需要注意，临时表的最大好处是它是线程安全的。

第二个存储过程，基于第一个存储过程生成的临时表而创建游标，并处理具体的工作。

第三个存储过程，作为一个入口，负责依次调用存储过程1和存储过程2，并提供相应的参数。

三个存储过程综合起来，就得到下面的例子：

2. CREATE PROCEDURE `proc1`(SourceDBName CHAR(50), SourceTableName CHAR(50))
3. BEGIN
4. DECLARE SQLStmt TEXT;
6. SET SQL\_NOTES=0;
8. SET @SQLStmt = CONCAT('DROP TEMPORARY TABLE IF EXISTS tmp\_table\_name');
9. PREPARE Stmt FROM @SQLStmt;
10. EXECUTE Stmt;
11. DEALLOCATE PREPARE Stmt;
13. SET @SQLStmt = CONCAT('CREATE TEMPORARY TABLE tmp\_table\_name SELECT ... FROM ',
14. SourceDBName,'.',SourceTableName,' WHERE ... ');
15. PREPARE Stmt FROM @SQLStmt;
16. EXECUTE Stmt;
17. DEALLOCATE PREPARE Stmt;
18. END$$
20. CREATE PROCEDURE `proc2`(TargetDBName CHAR(50), TargetTemplateTableName CHAR(50))
21. BEGIN
22. DECLARE done INT DEFAULT 0;
23. DECLARE FieldValue CHAR(50);
24. DECLARE CursorSegment CURSOR FOR SELECT Period FROM tmp\_table\_name;
25. DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
27. OPEN CursorSegment;
28. REPEAT
29. FETCH CursorSegment INTO FieldValue;
30. IF NOT done THEN
31. ...
32. END IF;
33. UNTIL done END REPEAT;
34. CLOSE CursorSegment;
35. END$$
37. CREATE PROCEDURE `proc3`(SourceDBName CHAR(50), SourceTableName CHAR(50),
38. TargetDBName CHAR(50), TargetTemplateTableName CHAR(50))
39. BEGIN
40. CALL proc1(SourceDBName, SourceTableName);
41. CALL proc2(TargetDBName, TargetTemplateTableName);
42. END$$

补充：运行前需要把系统参数变量“sql\_notes”设置为0，否则proc1在DROP TABLE时会停下来。原因是

2. "SQL\_NOTES = {0 | 1}
3. If set to 1 (the default), warnings of Note level are recorded.
4. If set to 0, Note warnings are suppressed."