# mysql存储过程

## 存储过程详解.

[**https://blog.csdn.net/wpydaguan/article/details/40787625**](https://blog.csdn.net/wpydaguan/article/details/40787625)

记录MYSQL存储过程中的关键语法：

1. DELIMITER //  声明语句结束符，用于区分;
2. **CREATE** **PROCEDURE** demo\_in\_parameter(IN p\_in **int**)  声明存储过程
3. **BEGIN** .... **END**  存储过程开始和结束符号
4. **SET** @p\_in=1   变量赋值
5. **DECLARE** l\_int **int** unsigned **default** 4000000;  变量定义



10. **什么是mysql存储例程?**
11. 存储例程是存储在数据库服务器中的一组sql语句，通过在查询中调用一个指定的名称来执行这些sql语句命令。
12. **为什么要使用mysql存储过程？**
13. 我们都知道应用程序分为两种，一种是基于web，一种是基于桌面，他们都和数据库进行交互来完成数据的存取工作。假设现在有一种应用程序包含了这两 种，现在要修改其中的一个查询sql语句，那么我们可能要同时修改他们中对应的查询sql语句，当我们的应用程序很庞大很复杂的时候问题就出现这，不易维 护！另外把sql查询语句放在我们的web程序或桌面中很容易遭到sql注入的破坏。而存储例程正好可以帮我们解决这些问题。
14. 存储过程(stored **procedure**)、存储例程(store routine)、存储函数区别
15. Mysql存储例程实际包含了存储过程和存储函数，它们被统称为存储例程。
16. 其中存储过程主要完成在获取记录或插入记录或更新记录或删除记录，即完成**select** **insert** **delete** **update**等的工作。而存储函数只完成查询的工作，可接受输入参数并返回一个结果。
17. 创建mysql存储过程、存储函数
18. **create** **procedure** 存储过程名(参数)
19. 存储过程体
20. **create** **function** 存储函数名(参数)

23. 下面是存储过程的例子：
24. mysql> DELIMITER //
25. mysql> **CREATE** **PROCEDURE** proc1(**OUT** s **int**)
26. -> **BEGIN**
27. -> **SELECT** COUNT(\*) **INTO** s **FROM** user;
28. -> **END**
29. -> //
30. mysql> DELIMITER ;
31. 注：
32. （1）这里需要注意的是DELIMITER//和DELIMITER;两句，DELIMITER是分割符的意思，因为MySQL默认以";"为分隔 符，如果我们没有声明分割符，那么编译器会把存储过程当成SQL语句进行处理，则存储过程的编译过程会报错，所以要事先用DELIMITER关键字申明当 前段分隔符，这样MySQL才会将";"当做存储过程中的代码，不会执行这些代码，用完了之后要把分隔符还原。
33. （2）存储过程根据需要可能会有输入、输出、输入输出参数，这里有一个输出参数s，类型是**int**型，如果有多个参数用","分割开。
34. （3）过程体的开始与结束使用**BEGIN**与**END**进行标识。
35. 这样，我们的一个MySQL存储过程就完成了，是不是很容易呢?看不懂也没关系，接下来，我们详细的讲解。

38. (2). 声明分割符
40. 其实，关于声明分割符，上面的注解已经写得很清楚，不需要多说，只是稍微要注意一点的是：如果是用MySQL的Administrator管理工具时，可以直接创建，不再需要声明。
42. (3). 参数
43. MySQL存储过程的参数用在存储过程的定义，共有三种参数类型,IN,**OUT**,INOUT,形式如：
44. CREATEPROCEDURE 存储过程名([[IN |**OUT** |INOUT ] 参数名 数据类形...])
45. IN 输入参数:表示该参数的值必须在调用存储过程时指定，在存储过程中修改该参数的值不能被返回，为默认值
46. **OUT** 输出参数:该值可在存储过程内部被改变，并可返回
47. INOUT 输入输出参数:调用时指定，并且可被改变和返回

### 参数传递注意：in 可以直接传递数值，out 需要传递参数，inout 需要传递参数；

**例如：SET @ff = 1; CALL del\_s(@ff);**

**CREATE DEFINER=`root`@`%` PROCEDURE `del\_s`(INOUT s INT)**

**BEGIN**

**SELECT s;**

**SELECT COUNT(\*) INTO s FROM account;**

**SELECT s;**

**END$$**

1. Ⅰ. IN参数例子
2. 创建:
3. 1.  mysql > DELIMITER //
4. 2.  mysql > **CREATE** **PROCEDURE** demo\_in\_parameter(IN p\_in **int**)
5. 3.       -> **BEGIN**
6. 4.       -> **SELECT** p\_in;
7. 5.       -> **SET** p\_in=2;
8. 6.       -> **SELECT** p\_in;
9. 7.       -> **END**;
10. 8.       -> //
11. 9.  mysql > DELIMITER ;
13. 执行结果:
14. 1.  mysql > **SET** @p\_in=1;
15. 2.  mysql > CALL demo\_in\_parameter(@p\_in);
16. 3.  +------+
17. 4.  | p\_in |
18. 5.  +------+
19. 6.  |   1  |
20. 7.  +------+
21. 8.
22. 9.  +------+
23. 10.| p\_in |
24. 11.+------+
25. 12.|   2  |
26. 13.+------+
27. 14.
28. 15.mysql> **SELECT** @p\_in;
29. 16.+-------+
30. 17.| @p\_in |
31. 18.+-------+
32. 19.|  1    |
33. 20.+-------+
35. 以上可以看出，p\_in虽然在存储过程中被修改，但并不影响@p\_id的值
37. Ⅱ.**OUT**参数例子
38. 创建:
39. 1.  mysql > DELIMITER //
40. 2.  mysql > **CREATE** **PROCEDURE** demo\_out\_parameter(**OUT** p\_out **int**)
41. 3.       -> **BEGIN**
42. 4.       -> **SELECT** p\_out;
43. 5.       -> **SET** p\_out=2;
44. 6.       -> **SELECT** p\_out;
45. 7.       -> **END**;
46. 8.       -> //
47. 9.  mysql > DELIMITER ;
49. 执行结果:
50. 1.  mysql > **SET** @p\_out=1;
51. 2.  mysql > CALL sp\_demo\_out\_parameter(@p\_out);
52. 3.  +-------+
53. 4.  | p\_out |
54. 5.  +-------+
55. 6.  | NULL  |
56. 7.  +-------+
57. 8.
58. 9.  +-------+
59. 10.| p\_out |
60. 11.+-------+
61. 12.|   2   |
62. 13.+-------+
63. 14.
64. 15.mysql> **SELECT** @p\_out;
65. 16.+-------+
66. 17.| p\_out |
67. 18.+-------+
68. 19.|   2   |
69. 20.+-------+
71. Ⅲ. INOUT参数例子
72. 创建:
73. 1.  mysql > DELIMITER //
74. 2.  mysql > **CREATE** **PROCEDURE** demo\_inout\_parameter(INOUT p\_inout **int**)
75. 3.       -> **BEGIN**
76. 4.       -> **SELECT** p\_inout;
77. 5.       -> **SET** p\_inout=2;
78. 6.       -> **SELECT** p\_inout;
79. 7.       -> **END**;
80. 8.       -> //
81. 9.  mysql > DELIMITER ;

84. 执行结果:
85. 1.  mysql > **SET** @p\_inout=1;
86. 2.  mysql > CALL demo\_inout\_parameter(@p\_inout) ;
87. 3.  +---------+
88. 4.  | p\_inout |
89. 5.  +---------+
90. 6.  |    1    |
91. 7.  +---------+
92. 8.
93. 9.  +---------+
94. 10.| p\_inout |
95. 11.+---------+
96. 12.|    2    |
97. 13.+---------+
98. 14.
99. 15.mysql > **SELECT** @p\_inout;
100. 16.+----------+
101. 17.| @p\_inout |
102. 18.+----------+
103. 19.|    2     |
104. 20.+----------+
106. (4). 变量
107. Ⅰ. 变量定义
108. 局部变量声明一定要放在存储过程体的开始
109. **DECLARE** variable\_name [,variable\_name...] datatype [**DEFAULT** value];
110. 其中，datatype为MySQL的数据类型，如:**int**, **float**, **date**,**varchar**(length)
111. 例如:
112. 1.  **DECLARE** l\_int **int** unsigned **default** 4000000;
113. 2.  **DECLARE** l\_numeric number(8,2) **DEFAULT** 9.95;
114. 3.  **DECLARE** l\_date **date** **DEFAULT** '1999-12-31';
115. 4.  **DECLARE** l\_datetime datetime **DEFAULT** '1999-12-31 23:59:59';
116. 5.  **DECLARE** l\_varchar **varchar**(255) **DEFAULT** 'This will not be padded';

119. Ⅱ. 变量赋值
120. **SET** 变量名 = 表达式值 [,variable\_name = expression ...]
122. Ⅲ. 用户变量
124. ⅰ. 在MySQL客户端使用用户变量
125. 1.  mysql > **SELECT** 'Hello World' **into** @x;
126. 2.  mysql > **SELECT** @x;
127. 3.  +-------------+
128. 4.  |   @x        |
129. 5.  +-------------+
130. 6.  | Hello World |
131. 7.  +-------------+
132. 8.  mysql > **SET** @y='Goodbye Cruel World';
133. 9.  mysql > **SELECT** @y;
134. 10.+---------------------+
135. 11.|     @y              |
136. 12.+---------------------+
137. 13.| Goodbye Cruel World |
138. 14.+---------------------+
139. 15.
140. 16.mysql > **SET** @z=1+2+3;
141. 17.mysql > **SELECT** @z;
142. 18.+------+
143. 19.| @z   |
144. 20.+------+
145. 21.|  6   |
146. 22.+------+
147. ⅱ. 在存储过程中使用用户变量
148. 1.  mysql > **CREATE** **PROCEDURE** GreetWorld( ) **SELECT** CONCAT(@greeting,' World');
149. 2.  mysql > **SET** @greeting='Hello';
150. 3.  mysql > CALL GreetWorld( );
151. 4.  +----------------------------+
152. 5.  | CONCAT(@greeting,' World') |
153. 6.  +----------------------------+
154. 7.  |  Hello World               |
155. 8.  +----------------------------+
156. ⅲ. 在存储过程间传递全局范围的用户变量
157. 1.  mysql> **CREATE** **PROCEDURE** p1()   **SET** @last\_procedure='p1';
158. 2.  mysql> **CREATE** **PROCEDURE** p2() **SELECT** CONCAT('Last procedure was ',@last\_procedure);
159. 3.  mysql> CALL p1( );
160. 4.  mysql> CALL p2( );
161. 5.  +-----------------------------------------------+
162. 6.  | CONCAT('Last procedure was ',@last\_proc       |
163. 7.  +-----------------------------------------------+
164. 8.  | **Last** **procedure** was p1                         |
165. 9.  +-----------------------------------------------+

168. 注意:
169. ①用户变量名一般以@开头
170. ②滥用用户变量会导致程序难以理解及管理
172. (5). 注释
174. MySQL存储过程可使用两种风格的注释
175. 双模杠：--
176. 该风格一般用于单行注释
177. c风格： 一般用于多行注释
178. 例如：
180. 1.  mysql > DELIMITER //
181. 2.  mysql > **CREATE** **PROCEDURE** proc1 --name存储过程名
182. 3.       -> (IN parameter1 **INTEGER**)
183. 4.       -> **BEGIN**
184. 5.       -> **DECLARE** variable1 **CHAR**(10);
185. 6.       -> IF parameter1 = 17 **THEN**
186. 7.       -> **SET** variable1 = 'birds';
187. 8.       -> **ELSE**
188. 9.       -> **SET** variable1 = 'beasts';
189. 10.     -> **END** IF;
190. 11.     -> **INSERT** **INTO** table1 **VALUES** (variable1);
191. 12.     -> **END**
192. 13.     -> //
193. 14.mysql > DELIMITER ;
195. 4.      MySQL存储过程的调用
196. 用call和你过程名以及一个括号，括号里面根据需要，加入参数，参数包括输入参数、输出参数、输入输出参数。具体的调用方法可以参看上面的例子。
197. 5.      MySQL存储过程的查询
198. 我们像知道一个数据库下面有那些表，我们一般采用showtables;进行查看。那么我们要查看某个数据库下面的存储过程，是否也可以采用呢？答案是，我们可以查看某个数据库下面的存储过程，但是是令一钟方式。
199. 我们可以用
200. selectname **from** mysql.proc **where** db=’数据库名’;
201. 或者
202. selectroutine\_name **from** information\_schema.routines **where** routine\_schema='数据库名';
203. 或者
204. showprocedure status **where** db='数据库名';
205. 进行查询。
206. 如果我们想知道，某个存储过程的详细，那我们又该怎么做呢？是不是也可以像操作表一样用describe 表名进行查看呢？
207. 答案是：我们可以查看存储过程的详细，但是需要用另一种方法：
208. SHOWCREATE **PROCEDURE** 数据库.存储过程名;
209. 就可以查看当前存储过程的详细。
211. 6.      MySQL存储过程的修改
212. **ALTER** **PROCEDURE**
213. 更改用**CREATE** **PROCEDURE** 建立的预先指定的存储过程，其不会影响相关存储过程或存储功能。
215. 7.      MySQL存储过程的删除
216. 删除一个存储过程比较简单，和删除表一样：
217. **DROP** **PROCEDURE**
218. 从MySQL的表格中删除一个或多个存储过程。
220. 8.      MySQL存储过程的控制语句
221. (1). 变量作用域
222. 内部的变量在其作用域范围内享有更高的优先权，当执行到**end**。变量时，内部变量消失，此时已经在其作用域外，变量不再可见了，应为在存储
223. 过程外再也不能找到这个申明的变量，但是你可以通过**out**参数或者将其值指派
224. 给会话变量来保存其值。

227. 1.  mysql > DELIMITER //
228. 2.  mysql > **CREATE** **PROCEDURE** proc3()
229. 3.       -> **begin**
230. 4.       -> **declare** x1 **varchar**(5) **default** 'outer';
231. 5.       -> **begin**
232. 6.       -> **declare** x1 **varchar**(5) **default** 'inner';
233. 7.       -> **select** x1;
234. 8.       -> **end**;
235. 9.       -> **select** x1;
236. 10.     -> **end**;
237. 11.     -> //
238. 12.mysql > DELIMITER ;
240. (2). 条件语句
241. Ⅰ. if-**then** -**else**语句
242. 1.  mysql > DELIMITER //
243. 2.  mysql > **CREATE** **PROCEDURE** proc2(IN parameter **int**)
244. 3.       -> **begin**
245. 4.       -> **declare** var **int**;
246. 5.       -> **set** var=parameter+1;
247. 6.       -> if var=0 **then**
248. 7.       -> **insert** **into** t **values**(17);
249. 8.       -> **end** if;
250. 9.       -> if parameter=0 **then**
251. 10.     -> **update** t **set** s1=s1+1;
252. 11.     -> **else**
253. 12.     -> **update** t **set** s1=s1+2;
254. 13.     -> **end** if;
255. 14.     -> **end**;
256. 15.     -> //
257. 16.mysql > DELIMITER ;
259. Ⅱ. case语句：
260. 1.  mysql > DELIMITER //
261. 2.  mysql > **CREATE** **PROCEDURE** proc3 (in parameter **int**)
262. 3.       -> **begin**
263. 4.       -> **declare** var **int**;
264. 5.       -> **set** var=parameter+1;
265. 6.       -> case var
266. 7.       -> **when** 0 **then**
267. 8.       -> **insert** **into** t **values**(17);
268. 9.       -> **when** 1 **then**
269. 10.     -> **insert** **into** t **values**(18);
270. 11.     -> **else**
271. 12.     -> **insert** **into** t **values**(19);
272. 13.     -> **end** case;
273. 14.     -> **end**;
274. 15.     -> //
275. 16.mysql > DELIMITER ;
276. case
277. **when** var=0 **then**
278. **insert** **into** t **values**(30);
279. **when** var>0 **then**
280. **when** var<0 **then**
281. **else**
283. **end** case


287. (3). 循环语句
288. Ⅰ. while ···· **end** while：
289. 1.  mysql > DELIMITER //
290. 2.  mysql > **CREATE** **PROCEDURE** proc4()
291. 3.       -> **begin**
292. 4.       -> **declare** var **int**;
293. 5.       -> **set** var=0;
294. 6.       -> while var<6 do
295. 7.       -> **insert** **into** t **values**(var);
296. 8.       -> **set** var=var+1;
297. 9.       -> **end** while;
298. 10.     -> **end**;
299. 11.     -> //
300. 12.mysql > DELIMITER ;
301. while条件 do
302. --循环体
303. endwhile

306. Ⅱ. repeat···· **end** repeat：
307. 它在执行操作后检查结果，而while则是执行前进行检查。
308. 1.  mysql > DELIMITER //
309. 2.  mysql > **CREATE** **PROCEDURE** proc5 ()
310. 3.       -> **begin**
311. 4.       -> **declare** v **int**;
312. 5.       -> **set** v=0;
313. 6.       -> repeat
314. 7.       -> **insert** **into** t **values**(v);
315. 8.       -> **set** v=v+1;
316. 9.       -> until v>=5
317. 10.     -> **end** repeat;
318. 11.     -> **end**;
319. 12.     -> //
320. 13.mysql > DELIMITER ;
321. repeat
322. --循环体
323. until循环条件
324. endrepeat;

327. Ⅲ. loop ·····endloop:
328. loop循环不需要初始条件，这点和while 循环相似，同时和repeat循环一样不需要结束条件, leave语句的意义是离开循环。
329. 1.  mysql > DELIMITER //
330. 2.  mysql > **CREATE** **PROCEDURE** proc6 ()
331. 3.       -> **begin**
332. 4.       -> **declare** v **int**;
333. 5.       -> **set** v=0;
334. 6.       -> LOOP\_LABLE:loop
335. 7.       -> **insert** **into** t **values**(v);
336. 8.       -> **set** v=v+1;
337. 9.       -> if v >=5 **then**
338. 10.     -> leave LOOP\_LABLE;
339. 11.     -> **end** if;
340. 12.     -> **end** loop;
341. 13.     -> **end**;
342. 14.     -> //
343. 15.mysql > DELIMITER ;

346. Ⅳ. LABLES 标号：
347. 标号可以用在**begin** repeat while 或者loop 语句前，语句标号只能在合法的语句前面使用。可以跳出循环，使运行指令达到复合语句的最后一步。
349. (4). ITERATE迭代
350. Ⅰ. ITERATE:
351. 1.    通过引用复合语句的标号,来从新开始复合语句
352. 2.  mysql > DELIMITER //
353. 3.  mysql > **CREATE** **PROCEDURE** proc10 ()
354. 4.       -> **begin**
355. 5.       -> **declare** v **int**;
356. 6.       -> **set** v=0;
357. 7.       -> LOOP\_LABLE:loop
358. 8.      -> if v=3 **then**
359. 9.      -> **set** v=v+1;
360. 10.        -> ITERATE LOOP\_LABLE;
361. 11.        -> **end** if;
362. 12.     -> **insert** **into** t **values**(v);
363. 13.     -> **set** v=v+1;
364. 14.        -> if v>=5 **then**
365. 15.        -> leave LOOP\_LABLE;
366. 16.     -> **end** if;
367. 17.     -> **end** loop;
368. 18.     -> **end**;
369. 19.     -> //
370. 20.mysql > DELIMITER ;

373. 9.      MySQL存储过程的基本函数
375. (1).字符串类
376. CHARSET(str) //返回字串字符集
377. CONCAT (string2 [,... ]) //连接字串
378. INSTR (string ,substring ) //返回substring首次在string中出现的位置,不存在返回0
379. LCASE (string2 ) //转换成小写
380. LEFT (string2 ,length ) //从string2中的左边起取length个字符
381. LENGTH (string ) //string长度
382. LOAD\_FILE (file\_name ) //从文件读取内容
383. LOCATE (substring , string [,start\_position ] ) 同INSTR,但可指定开始位置
384. LPAD (string2 ,length ,pad ) //重复用pad加在string开头,直到字串长度为length
385. LTRIM (string2 ) //去除前端空格
386. REPEAT (string2 ,count ) //重复count次
387. REPLACE (str ,search\_str ,replace\_str ) //在str中用replace\_str替换search\_str
388. RPAD (string2 ,length ,pad) //在str后用pad补充,直到长度为length
389. RTRIM (string2 ) //去除后端空格
390. STRCMP (string1 ,string2 ) //逐字符比较两字串大小,
391. SUBSTRING (str , position [,length ]) //从str的position开始,取length个字符,
392. 注：mysql中处理字符串时，默认第一个字符下标为1，即参数position必须大于等于1
393. 1.  mysql> **select** substring('abcd',0,2);
394. 2.  +-----------------------+
395. 3.  | substring('abcd',0,2) |
396. 4.  +-----------------------+
397. 5.  |                       |
398. 6.  +-----------------------+
399. 7.  1 row in **set** (0.00 sec)
400. 8.
401. 9.  mysql> **select** substring('abcd',1,2);
402. 10.+-----------------------+
403. 11.| substring('abcd',1,2) |
404. 12.+-----------------------+
405. 13.|     ab                |
406. 14.+-----------------------+
407. 15.1 row in **set** (0.02 sec)
408. TRIM([[BOTH|LEADING|TRAILING][padding] **FROM**]string2) //去除指定位置的指定字符
409. UCASE (string2 ) //转换成大写
410. RIGHT(string2,length) //取string2最后length个字符
411. SPACE(count) //生成count个空格
412. (2).数学类
413. ABS (number2 ) //绝对值
414. BIN (decimal\_number ) //十进制转二进制
415. CEILING (number2 ) //向上取整
416. CONV(number2,from\_base,to\_base) //进制转换
417. FLOOR (number2 ) //向下取整
418. FORMAT (number,decimal\_places ) //保留小数位数
419. HEX (DecimalNumber ) //转十六进制
420. 注：HEX()中可传入字符串，则返回其**ASC**-11码，如HEX('DEF')返回4142143
421. 也可以传入十进制整数，返回其十六进制编码，如HEX(25)返回19
422. LEAST (number , number2 [,..]) //求最小值
423. MOD (numerator ,denominator ) //求余
424. POWER (number ,power ) //求指数
425. RAND([seed]) //随机数
426. ROUND (number [,decimals ]) //四舍五入,decimals为小数位数]
427. 注：返回类型并非均为整数，如：
428. (1)默认变为整形值
429. 1.  mysql> **select** round(1.23);
430. 2.  +-------------+
431. 3.  | round(1.23) |
432. 4.  +-------------+
433. 5.  |           1 |
434. 6.  +-------------+
435. 7.  1 row in **set** (0.00 sec)
436. 8.
437. 9.  mysql> **select** round(1.56);
438. 10.+-------------+
439. 11.| round(1.56) |
440. 12.+-------------+
441. 13.|           2 |
442. 14.+-------------+
443. 15.1 row in **set** (0.00 sec)

446. (2)可以设定小数位数，返回浮点型数据
447. 1.  mysql> **select** round(1.567,2);
448. 2.  +----------------+
449. 3.  | round(1.567,2) |
450. 4.  +----------------+
451. 5.  |           1.57 |
452. 6.  +----------------+
453. 7.  1 row in **set** (0.00 sec)
454. SIGN (number2 ) //
456. (3).日期时间类
457. ADDTIME (date2 ,time\_interval )//将time\_interval加到date2
458. CONVERT\_TZ (datetime2 ,fromTZ ,toTZ ) //转换时区
459. **CURRENT\_DATE** ( ) //当前日期
460. **CURRENT\_TIME** ( ) //当前时间
461. CURRENT\_TIMESTAMP ( ) //当前时间戳
462. **DATE** (datetime ) //返回datetime的日期部分
463. DATE\_ADD (date2 , INTERVAL d\_value d\_type ) //在date2中加上日期或时间
464. DATE\_FORMAT (datetime ,FormatCodes ) //使用formatcodes格式显示datetime
465. DATE\_SUB (date2 , INTERVAL d\_value d\_type ) //在date2上减去一个时间
466. DATEDIFF (date1 ,date2 ) //两个日期差
467. DAY (**date** ) //返回日期的天
468. DAYNAME (**date** ) //英文星期
469. DAYOFWEEK (**date** ) //星期(1-7) ,1为星期天
470. DAYOFYEAR (**date** ) //一年中的第几天
471. EXTRACT (interval\_name **FROM** **date** ) //从**date**中提取日期的指定部分
472. MAKEDATE (year ,day ) //给出年及年中的第几天,生成日期串
473. MAKETIME (**hour** ,**minute** ,**second** ) //生成时间串
474. MONTHNAME (**date** ) //英文月份名
475. NOW ( ) //当前时间
476. SEC\_TO\_TIME (seconds ) //秒数转成时间
477. STR\_TO\_DATE (string ,format ) //字串转成时间,以format格式显示
478. TIMEDIFF (datetime1 ,datetime2 ) //两个时间差
479. TIME\_TO\_SEC (**time** ) //时间转秒数]
480. WEEK (date\_time [,start\_of\_week ]) //第几周
481. YEAR (datetime ) //年份
482. DAYOFMONTH(datetime) //月的第几天
483. **HOUR**(datetime) //小时
484. LAST\_DAY(**date**) //**date**的月的最后日期
485. MICROSECOND(datetime) //微秒
486. MONTH(datetime) //月
487. **MINUTE**(datetime) //分返回符号,正负或0
488. SQRT(number2) //开平方




494. MySql分页存储过程
495. MySql测试版本：5.0.41-community-nt
497. **DROP** **PROCEDURE** IF EXISTS pr\_pager;
498. **CREATE** **PROCEDURE** pr\_pager(
500. IN    p\_table\_name        **VARCHAR**(1024),
501. IN    p\_fields            **VARCHAR**(1024),
502. IN    p\_page\_size            **INT**,
503. IN    p\_page\_now            **INT**,
504. IN    p\_order\_string        **VARCHAR**(128),
505. IN    p\_where\_string        **VARCHAR**(1024),
506. **OUT**    p\_out\_rows            **INT**
508. )
509. NOT DETERMINISTIC
510. SQL SECURITY DEFINER
511. COMMENT '分页存储过程'
513. **BEGIN**

516. **DECLARE** m\_begin\_row **INT** **DEFAULT** 0;
517. **DECLARE** m\_limit\_string **CHAR**(64);

520. **SET** m\_begin\_row = (p\_page\_now - 1) \* p\_page\_size;
521. **SET** m\_limit\_string = CONCAT(' LIMIT ', m\_begin\_row, ', ', p\_page\_size);
523. **SET** @COUNT\_STRING = CONCAT('SELECT COUNT(\*) INTO @ROWS\_TOTAL FROM ', p\_table\_name, ' ', p\_where\_string);
524. **SET** @MAIN\_STRING = CONCAT('SELECT ', p\_fields, ' FROM ', p\_table\_name, ' ', p\_where\_string, ' ', p\_order\_string,m\_limit\_string);

527. **PREPARE** count\_stmt **FROM** @COUNT\_STRING;  //预处理语句
528. **EXECUTE** count\_stmt;   //执行语句
529. **DEALLOCATE** **PREPARE** count\_stmt;   //释放资源
530. **SET** p\_out\_rows = @ROWS\_TOTAL;
532. **PREPARE** main\_stmt **FROM** @MAIN\_STRING;   //预处理语句
533. **EXECUTE** main\_stmt;   //执行语句
534. **DEALLOCATE** **PREPARE** main\_stmt;   //释放资源
536. **END**;

## 二.资料补充.

### 1. MySQL IF语句语法

下面说明了IF语句的语法：

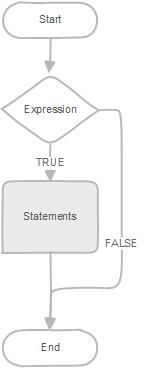
IF expression THEN

statements;

END IF;

如果表达式(expression)计算结果为TRUE，那么将执行statements语句，否则控制流将传递到END IF之后的下一个语句。

以下流程图演示了IF语句的执行过程：



### 2.MySQL IF ELSE语句

如果表达式计算结果为FALSE时执行语句，请使用IF ELSE语句，如下所示：

IF expression THEN

statements;

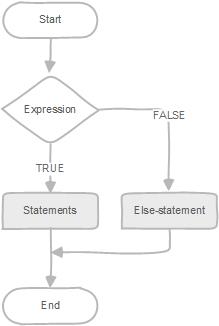
ELSE

else-statements;

END IF;

SQL

以下流程图说明了IF ELSE语句的执行过程：



### 3.MySQL IF ELSEIF ELSE语句

如果要基于多个表达式有条件地执行语句，则使用IF ELSEIF ELSE语句如下：

IF expression THEN

statements;

ELSEIF elseif-expression THEN

elseif-statements;

...

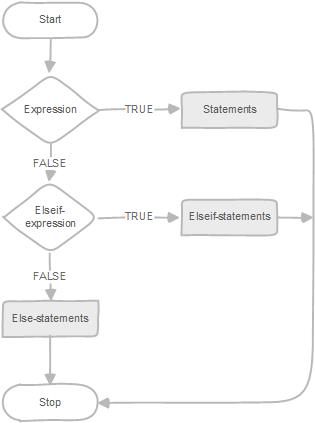
ELSE

else-statements;

END IF;

SQL

如果表达式(expression)求值为TRUE，则IF分支中的语句(statements)将执行；如果表达式求值为FALSE，则如果elseif\_expression的计算结果为TRUE，MySQL将执行elseif-expression，否则执行ELSE分支中的else-statements语句。具体流程如下 -



### 4.MySQL IF语句示例

以下示例说明如何使用IF ESLEIF ELSE语句，GetCustomerLevel()存储过程接受客户编号和客户级别的两个参数。

首先，它从customers表中获得信用额度

然后，根据信用额度，它决定客户级别：PLATINUM , GOLD 和 SILVER 。

参数p\_customerlevel存储客户的级别，并由调用程序使用。

USE yiibaidb;

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(

in p\_customerNumber int(11),

out p\_customerLevel varchar(10))

BEGIN

DECLARE creditlim double;

SELECT creditlimit INTO creditlim

FROM customers

WHERE customerNumber = p\_customerNumber;

IF creditlim > 50000 THEN

SET p\_customerLevel = 'PLATINUM';

ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN

SET p\_customerLevel = 'GOLD';

ELSEIF creditlim < 10000 THEN

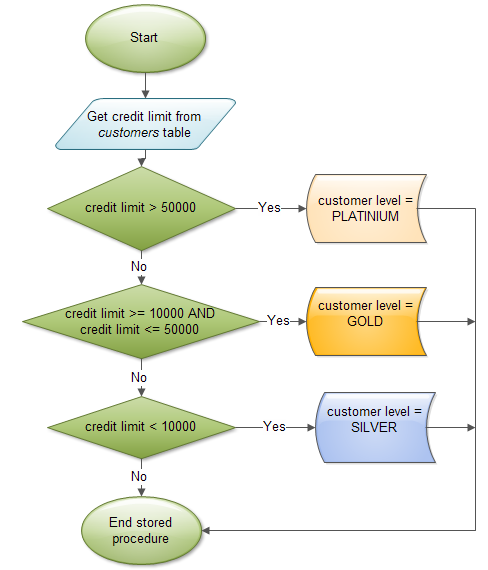
SET p\_customerLevel = 'SILVER';

END IF;

END$$

SQL

以下流程图演示了确定客户级别的逻辑 -



在本教程中，您已经学会了如何使用MySQL IF语句根据条件执行一个SQL代码块。

### 5. SELECT INTO 语句 (mysql不支持但在存储过程中支持)

SELECT INTO 语句从一个表中选取数据，然后把数据插入另一个表中。

SELECT INTO 语句常用于创建表的备份复件或者用于对记录进行存档。

### 6.SQL SELECT INTO 语法

您可以把所有的列插入新表：

SELECT \*

INTO new\_table\_name [IN externaldatabase]

FROM old\_tablename

或者只把希望的列插入新表：

SELECT column\_name(s)

INTO new\_table\_name [IN externaldatabase]

FROM old\_tablename

### 7.SQL SELECT INTO 实例 - 制作备份复件

下面的例子会制作 "Persons" 表的备份复件：

SELECT \*

INTO Persons\_backup

FROM Persons

IN 子句可用于向另一个数据库中拷贝表：

SELECT \*

INTO Persons IN 'Backup.mdb'

FROM Persons

如果我们希望拷贝某些域，可以在 SELECT 语句后列出这些域：

SELECT LastName,FirstName

INTO Persons\_backup

FROM Persons

### 8.SQL SELECT INTO 实例 - 带有 WHERE 子句

我们也可以添加 WHERE 子句。

下面的例子通过从 "Persons" 表中提取居住在 "Beijing" 的人的信息，创建了一个带有两个列的名为 "Persons\_backup" 的表：

SELECT LastName,Firstname

INTO Persons\_backup

FROM Persons

WHERE City='Beijing

### 9.Mysql支持SQL INSERT INTO SELECT 语句

SQL INSERT INTO SELECT 语法

我们可以从一个表中复制所有的列插入到另一个已存在的表中：

INSERT INTO table2  
SELECT \* FROM table1;

或者我们可以只复制希望的列插入到另一个已存在的表中：

INSERT INTO table2  
(column\_name(s))  
SELECT column\_name(s)  
FROM table1;

## 三.存储过程调用

### 1.调用过程

org.springframework.jdbc.core.SimpleJdbcCall类是表示对存储过程或存储函数的调用的多线程，可重用的对象。 它提供元数据处理以简化访问基本存储过程/函数所需的代码。 所有需要提供的是程序/函数的名称和包含执行调用时参数的Map对象。 提供的参数的名称将与创建存储过程时声明的输入和输出参数相匹配。

使用到的 Student 表的结构如下 -

CREATE TABLE Student(

ID INT NOT NULL AUTO\_INCREMENT,

NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL,

PRIMARY KEY (ID)

);

SQL

**类的声明**

以下是org.springframework.jdbc.core.SimpleJdbcCall接口的声明 -

public class SimpleJdbcCall

extends AbstractJdbcCall

implements SimpleJdbcCallOperations

Java

以下示例将演示如何使用Spring的SimpleJdbcCall对象调用存储过程。通过调用存储过程来读取Student表中的一条记录。传递一个学生编号作为参数并读取学生记录。

**语法**

SimpleJdbcCall jdbcCall = new SimpleJdbcCall(dataSource).withProcedureName("getRecord");

SqlParameterSource in = new MapSqlParameterSource().addValue("in\_id", id);

Map<String, Object> out = jdbcCall.execute(in);

Student student = new Student();

student.setId(id);

student.setName((String) out.get("out\_name"));

student.setAge((Integer) out.get("out\_age"));

Java

在上代码中，

* *jdbcCall* - SimpleJdbcCall对象来表示存储过程。
* *in* - SqlParameterSource对象将参数传递给存储过程。
* *student* - 学生(Student)对象。
* *out* - Map对象来表示存储过程调用结果的输出。

### 2.在Spring中有许多方法来调用存储过程。

* 如果使用CallableStatementCreator来声明参数，则将使用Java的标准接口CallableStatement，即注册输出参数并单独设置它们。使用SqlParameter抽象将使你的代码更清洁。
* 我建议你看SimpleJdbcCall。它可以这样使用：
* SimpleJdbcCall jdbcCall = new SimpleJdbcCall(jdbcTemplate)
* .withSchemaName(schema)
* .withCatalogName(package)
* .withProcedureName(procedure)();
* ...
* jdbcCall.addDeclaredParameter(new SqlParameter(paramName, OracleTypes.NUMBER));
* ...
* jdbcCall.execute(callParams);
* 对于简单的过程，你可以使用jdbcTemplate的更新方法：
* jdbcTemplate.update("call SOME\_PROC (?, ?)", param1, param2);

**存储过程：**

* DELIMITER $$
* DROP PROCEDURE IF EXISTS `TEST`.`getRecord` $$
* CREATE PROCEDURE `TEST`.`getRecord` (
* IN in\_id INTEGER,
* OUT out\_name VARCHAR(20),
* OUT out\_age INTEGER)
* BEGIN
* SELECT name, age
* INTO out\_name, out\_age
* FROM Student where id = in\_id;
* END $$
* DELIMITER ;

### 使用案例

public void deleteAccount(String id) {

String call ="del\_account"; //存储过程名称

SimpleJdbcCall jdbcCall = new SimpleJdbcCall(jdbcTemplate); //设置数据源

jdbcCall.withProcedureName(call); //设置名称

MapSqlParameterSource parameter = new MapSqlParameterSource();

parameter.addValue("temp\_id",id); //设置in参数

jdbcCall.execute(parameter); //传入参数执行

}

**存储过程：**

**DELIMITER $$**

**USE `unilab2018`$$**

**DROP PROCEDURE IF EXISTS `del\_account`$$**

**CREATE DEFINER=`root`@`%` PROCEDURE `del\_account`(IN temp\_id VARCHAR(64))**

**BEGIN**

**DELETE FROM account\_team WHERE account\_id = temp\_id;**

**DELETE FROM accoun\_role WHERE account\_id = temp\_id;**

**DELETE FROM account WHERE id = temp\_id;**

**END$$**

**DELIMITER ;**

## 四.预处理语句

[MySQL 预处理语句prepare、execute、deallocate的使用](http://blog.itpub.net/29773961/viewspace-1852824/)

MySQL官方将prepare、execute、deallocate统称为PREPARE STATEMENT。  
我习惯称其为【预处理语句】。  
  
其用法十分简单，

1. **PREPARE stmt\_name FROM preparable\_stmt**
2. **EXECUTE stmt\_name**
3. **[USING @var\_name** **[,** **@var\_name]** **...]    -**
4. **{DEALLOCATE |** **DROP} PREPARE stmt\_name**

举个栗子：

1. mysql> PREPARE pr1 FROM 'SELECT ?+?';
2. Query OK, 0 rows affected (0.01 sec)
3. Statement prepared
4. mysql> SET @a=1, @b=10 ;
5. Query OK, 0 rows affected (0.00 sec)
6. mysql> EXECUTE pr1 USING @a, @b;
7. +------+
8. | ?+?  |
9. +------+
10. | 11   |
11. +------+
12. 1 row in set (0.00 sec)
14. mysql> EXECUTE pr1 USING 1, 2;    -- 只能使用用户变量传递。
15. ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the
16. right syntax to use near '1, 2' at line 1
17. mysql> DEALLOCATE PREPARE pr1;
18. Query OK, 0 rows affected (0.00 sec)

使用PAREPARE STATEMENT可以减少每次执行SQL的语法分析，  
比如用于执行带有WHERE条件的SELECT和DELETE，或者UPDATE，或者INSERT，只需要每次修改变量值即可。  
同样可以防止SQL注入，参数值可以包含转义符和定界符。  
  
适用在应用程序中，或者SQL脚本中均可。  
  
更多用法：  
  
同样PREPARE ... FROM可以直接接用户变量：

1. mysql> CREATE TABLE a (a int);
2. Query OK, 0 rows affected (0.26 sec)
3. mysql> INSERT INTO a SELECT 1;
4. Query OK, 1 row affected (0.04 sec)
5. Records: 1 Duplicates: 0 Warnings: 0
6. mysql> INSERT INTO a SELECT 2;
7. Query OK, 1 row affected (0.04 sec)
8. Records: 1 Duplicates: 0 Warnings: 0
9. mysql> INSERT INTO a SELECT 3;
10. Query OK, 1 row affected (0.04 sec)
11. Records: 1 Duplicates: 0 Warnings: 0
12. mysql> SET @select\_test = CONCAT('SELECT \* FROM ', @table\_name);
13. Query OK, 0 rows affected (0.00 sec)
14. mysql> SET @table\_name = 'a';
15. Query OK, 0 rows affected (0.00 sec)
16. mysql> PREPARE pr2 FROM @select\_test;
17. Query OK, 0 rows affected (0.00 sec)
18. Statement prepared
19. mysql> EXECUTE pr2 ;
20. +------+
21. | a    |
22. +------+
23. | 1    |
24. | 2    |
25. | 3    |
26. +------+
27. 3 rows in set (0.00 sec)
28. mysql> DROP PREPARE pr2;    -- 此处DROP可以替代DEALLOCATE
29. Query OK, 0 rows affected (0.00 sec)

每一次执行完EXECUTE时，养成好习惯，须执行DEALLOCATE PREPARE … 语句，这样可以释放执行中使用的所有数据库资源（如游标）。  
不仅如此，如果一个session的预处理语句过多，可能会达到max\_prepared\_stmt\_count的上限值。  
  
预处理语句只能在创建者的会话中可以使用，其他会话是无法使用的。  
而且在任意方式（正常或非正常）退出会话时，之前定义好的预处理语句将不复存在。  
如果在存储过程中使用，如果不在过程中DEALLOCATE掉，在存储过程结束之后，该预处理语句仍然会有效。

## 五.存储过程游标

### 1.游标的作用及属性

游标的作用就是用于对查询数据库所返回的记录进行遍历，以便进行相应的操作；游标有下面这些属性：

a、游标是只读的，也就是不能更新它；

b、游标是不能滚动的，也就是只能在一个方向上进行遍历，不能在记录之间随意进退，不能跳过某些记录；

c、避免在已经打开游标的表上更新数据。

### 2.如何使用游标(cursor)

**1.声明游标**

DECLARE cursor\_name CURSOR FOR select\_statement  
这个语句声明一个游标。也可以在子程序中定义多个游标，但是一个块中的每一个游标必须有唯一的名字。声明游标后也是单条操作的，但是不能用SELECT语句不能有INTO子句。

**2. 游标OPEN语句**

OPEN cursor\_name  
这个语句打开先前声明的游标。

**3. 游标FETCH语句**

FETCH cursor\_name INTO var\_name [, var\_name] ...  
这个语句用指定的打开游标读取下一行（如果有下一行的话），并且前进游标指针。

**4. 游标CLOSE语句**

CLOSE cursor\_name  
这个语句关闭先前打开的游标。

### 3.使用实例.

**EG01：**

**DELIMITER $$**

**USE `scopus` $$**

**DROP PROCEDURE IF EXISTS `sp\_author\_doc\_num\_year`$$**

**CREATE DEFINER=`root`@`127.0.0.1` PROCEDURE `sp\_author\_doc\_num\_year`(IN in\_authorId INT)**

**label:BEGIN**

**DECLARE preYear INT;**

**DECLARE pubyear\_ SMALLINT;**

**DECLARE num\_ INT;**

**-- 遍历数据结束标志**

**DECLARE done INT DEFAULT FALSE;**

**-- 游标**

**DECLARE cur CURSOR FOR SELECT pubyear,num FROM dso\_author\_doc\_num\_year WHERE authorid=in\_authorId ORDER BY pubyear;**

**-- 将结束标志绑定到游标**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;**

**DROP TABLE IF EXISTS t\_item;**

**CREATE TEMPORARY TABLE t\_item(**

**pubyear SMALLINT NOT NULL,**

**num INT NOT NULL**

**);**

**SET preYear = 0;**

**-- 打开游标**

**OPEN cur;**

**-- 开始循环**

**read\_loop: LOOP**

**-- 提取游标里的数据，这里只有一个，多个的话也一样；**

**FETCH cur INTO pubyear\_,num\_;**

**-- 声明结束的时候**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**IF preYear=0 THEN**

**SET preYear = pubyear\_+1;**

**INSERT INTO t\_item VALUES(pubyear\_,num\_);**

**ELSE**

**WHILE pubyear\_-preYear >= 0 DO**

**IF pubyear\_-preYear > 0 THEN**

**INSERT INTO t\_item VALUES(preyear,0);**

**ELSE**

**INSERT INTO t\_item VALUES(pubyear\_,num\_);**

**END IF;**

**SET preYear = preYear + 1;**

**END WHILE;**

**END IF;**

**END LOOP;**

**-- 关闭游标**

**CLOSE cur;**

**SELECT \* FROM t\_item ORDER BY pubyear;**

**-- if maxYear=0 THEN leave label;**

**-- SELECT maxYear;**

**END**

**;$$**

**DELIMITER ;**

**EG2:**

DELIMITER $$

USE `scopus`$$

DROP PROCEDURE IF EXISTS `sp\_author\_source\_pie`$$

CREATE DEFINER=`root`@`127.0.0.1` PROCEDURE `sp\_author\_source\_pie`(IN in\_authorId INT)

label:BEGIN

-- 4186

-- DECLARE others int;

SET @maxpie = 6;

DROP TEMPORARY TABLE IF EXISTS t\_item;

CREATE TEMPORARY TABLE t\_item

SELECT authorid,sourceid,doc\_num,total\_doc\_num,ratio,sourceTitle

FROM dso\_author\_source WHERE authorid = in\_authorId ORDER BY ratio DESC;

SELECT COUNT(\*) INTO @pieNum FROM t\_item;

IF @pieNum <= @maxpie THEN

SELECT \* FROM t\_item ORDER BY sourceTitle;

LEAVE label;

END IF;

-- 取排名靠前的数据

DROP TEMPORARY TABLE IF EXISTS t\_item\_top;

CREATE TEMPORARY TABLE t\_item\_top

SELECT authorid,sourceid,doc\_num,total\_doc\_num,ratio,sourceTitle

FROM t\_item ORDER BY ratio DESC LIMIT 6; -- @maxpie

SELECT total\_doc\_num INTO @totalNum FROM t\_item LIMIT 1;

SELECT SUM(doc\_num) INTO @topNum FROM t\_item\_top;

INSERT INTO t\_item\_top(authorid,sourceid,doc\_num,total\_doc\_num,ratio,sourceTitle)

VALUES( in\_authorId,0,@totalNum-@topNum,@totalNum,(@totalNum-@topNum)/@totalNum,'others');

SELECT \* FROM t\_item\_top;

END

;$$

DELIMITER ;

**\*EG3:**

**DELIMITER $$**

**USE `scopus`$$**

**DROP PROCEDURE IF EXISTS `sp\_show\_scholar\_network`$$**

**CREATE DEFINER=`root`@`127.0.0.1` PROCEDURE `sp\_show\_scholar\_network`(IN authorid\_ INT, IN startYear\_ SMALLINT, IN endYear\_ SMALLINT, IN min\_num INT, IN max\_num INT, IN max\_distance INT)**

**mySP:BEGIN**

**-- 1,2012,2014,1,10,300**

**DECLARE source INT;**

**DECLARE target INT;**

**DECLARE num\_ SMALLINT;**

**DECLARE x\_ DOUBLE;**

**DECLARE y\_ DOUBLE;**

**DECLARE len\_ DOUBLE;**

**DECLARE degree\_ DOUBLE;**

**DECLARE i\_ INT;**

**DECLARE done INT DEFAULT FALSE;**

**DECLARE cur CURSOR FOR SELECT author\_source,author\_target,num FROM t\_item;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;**

**DROP TEMPORARY TABLE IF EXISTS t\_item;**

**CREATE TEMPORARY TABLE t\_item(**

**author\_source INT NOT NULL,**

**author\_target INT NOT NULL,**

**num SMALLINT,**

**X DOUBLE DEFAULT NULL,**

**Y DOUBLE DEFAULT NULL,**

**name\_en\_source VARCHAR(100),**

**name\_en\_target VARCHAR(100),**

**name\_ch\_source VARCHAR(50),**

**name\_ch\_target VARCHAR(50)**

**);**

**INSERT INTO t\_item(author\_source,author\_target,num,name\_en\_source,name\_en\_target,name\_ch\_source,name\_ch\_target)**

**(SELECT authorid1,authorid2,num,name1\_en,name2\_en,name1\_ch,name2\_ch FROM dso\_scholar\_network\_num\_year WHERE authorid1 = authorid\_**

**AND pubyear>=startYear\_ AND pubyear <= endYear\_ AND num >=min\_num AND num<=max\_num);**

**-- 计算x,y**

**SELECT COUNT(\*) INTO @num\_direct FROM t\_item; -- 直接关系人的数量**

**IF @num\_direct = 0 THEN**

**LEAVE mySP;**

**END IF;**

**SELECT MAX(num) INTO @max\_num\_ FROM t\_item; -- 最大联系数量**

**SET @degree = 2\*PI()/@num\_direct;**

**SET i\_ = 0;**

**OPEN cur;**

**read\_loop: LOOP**

**FETCH cur INTO source,target,num\_;**

**IF done THEN**

**LEAVE read\_loop;**

**END IF;**

**SET len\_ = max\_distance/@max\_num\_\*num\_;**

**SET degree\_ = @degree \* i\_;**

**SET x\_ = len\_ \* COS(degree\_);**

**SET y\_ = len\_ \* SIN(degree\_);**

**UPDATE t\_item SET X=x\_, Y=y\_ WHERE author\_source=source AND author\_target=target;**

**SET i\_ = i\_ + 1;**

**END LOOP;**

**-- 关闭游标**

**CLOSE cur;**

**SELECT GROUP\_CONCAT(author\_target SEPARATOR",") INTO @scholars FROM t\_item;**

**DROP TEMPORARY TABLE IF EXISTS t\_item2;**

**CREATE TEMPORARY TABLE t\_item2(**

**author\_source INT NOT NULL,**

**author\_target INT NOT NULL,**

**num SMALLINT NOT NULL,**

**X DOUBLE DEFAULT NULL,**

**Y DOUBLE DEFAULT NULL,**

**name\_en\_source VARCHAR(100),**

**name\_en\_target VARCHAR(100),**

**name\_ch\_source VARCHAR(50),**

**name\_ch\_target VARCHAR(50)**

**);**

**SET @sql =CONCAT('insert into t\_item2(author\_source,author\_target,num,name\_en\_source,name\_en\_target,name\_ch\_source,name\_ch\_target) ',**

**'(select authorid1,authorid2,num,name1\_en,name2\_en,name1\_ch,name2\_ch from dso\_scholar\_network\_num\_year where authorid1 in (',@scholars,**

**') and authorid2 in (',@scholars,') and pubyear>=', startYear\_, ' and pubyear <= ', endYear\_ ,')');**

**PREPARE stmt FROM @sql;**

**EXECUTE stmt;**

**DEALLOCATE PREPARE stmt;**

**SELECT \* FROM t\_item UNION SELECT \* FROM t\_item2;**

**END$$**

**DELIMITER ;**