**MySQL存储过程异常处理**

在使用MySQL存储过程时，其中的代码可能会出现运行错误从而导致异常，此时需要将存储过程中产生的异常捕获并打印出来

**需要知道的概念：**

* condition
* hanlder
* diagnostics area（诊断区）

**1. condition**

存储过程中出现的错误事件也就是异常都可以被称为condition。

**declare condition语法：**

DECLARE condition\_name CONDITION FOR condition\_value

condition\_value:

mysql\_error\_code

| SQLSTATE [VALUE] sqlstate\_value

* 1
* 2
* 3
* 4
* 5

declare condition语句的作用是给需要进行处理的condition定义一个名称，并提供给后续的declare handler进行调用，从而使代码清晰化。

例如：定义一个名称为"no\_such\_table"的condition，并在declare handler中调用该名称。

declare condition中可以使用error code(报错的代码)值或是sqlstate(5位的字符串)值。

使用error code值定义condition：

DECLARE no\_such\_table CONDITION FOR 1051;

DECLARE CONTINUE HANDLER FOR no\_such\_table

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4
* 5

使用sqlstate值定义condition

DECLARE no\_such\_table CONDITION FOR SQLSTATE '42S02';

DECLARE CONTINUE HANDLER FOR no\_such\_table

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4
* 5

开头为’0’的error code或是开头为’00’的sqlstate值不能用于定义condition，因为它们代表的是成功，而不是异常。

在SIGNAL或者是RESIGNAL中引用的condition，必须是使用sqlstate定义的condition，不能使用error code定义的condition。

存储过程中的declare condition语句，必须出现在declare cursor或是declare handler之前，否则会报错。

**2.handler**  
handler就是用来处理condition的，当定义的condition发生时，就执行handler中定义的处理逻辑，handler可以处理多个condition。

**declare handler语法：**

DECLARE handler\_action HANDLER

FOR condition\_value [, condition\_value] ...

statement

handler\_action:

CONTINUE

| EXIT

| UNDO

condition\_value:

mysql\_error\_code

| SQLSTATE [VALUE] sqlstate\_value

| condition\_name

| SQLWARNING

| NOT FOUND

| SQLEXCEPTION

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

declare handler的语句必须在declare condition语句和定义变量语句之后出现

当handler中定义的condition触发时，可以采取以下三种处理方式：  
1.CONTINUE  
发送错误时继续执行后续代码  
2.EXIT  
发生错误时退出该handler定义所在的代码块（可能是子代码块或者main代码块）  
3.UNDO  
回滚所有的操作，目前还不支持，所以只有continue和exit可用。

**示例：**

1.使用error code定义handler

DECLARE CONTINUE HANDLER FOR 1051

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4

2.使用sqlstate值定义handler

DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4

3.SQLWARNING代表01开头的sqlstate值

DECLARE CONTINUE HANDLER FOR SQLWARNING

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4

4.NOT FOUND代表02开头的sqlstate值，这通常用于具有游标的上下文关系中，用来处理游标走到数据集终点时的condition。

DECLARE CONTINUE HANDLER FOR NOT FOUND

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4

5.SQLEXCEPTION代表所有其他不是以00,01,02开头的sqlstate值

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

BEGIN

-- body of handler

END;

* 1
* 2
* 3
* 4

**注意1：**  
在存储过程中，如果出现了一个condition，但是此时没有定义相关的handler，那么处理该condition的方法取决于该condition的类型

SQLEXCEPTION类型的condition  
默认使用EXIT handler来进行处理，如果此时该存储过程被另外一个存储过程调用，那么将使用调用者中定义的handler来进行处理。

SQLWARNING类型的condition  
默认使用CONTINUE handler来进行处理，存储过程继续执行。

NOT FOUND类型的condition  
如果condition被正常抛出，那么存储过程正常执行，也就是continue的处理方式，如果是被SIGNAL或RESIGNAL抛出，那么存储过程终止运行，也就是exit的处理方式

来看官网的一个SQLSTATE '23000’主键冲突的例子：

mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()

-> BEGIN

-> DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;

-> SET @x = 1;

-> INSERT INTO test.t VALUES (1);

-> SET @x = 2;

-> INSERT INTO test.t VALUES (1);

-> SET @x = 3;

-> END;

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//

+------+

| @x |

+------+

| 3 |

+------+

1 row in set (0.00 sec)

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

可以看到存储过程是正常执行的。

如果希望被handler对捕获到condition不进行任何处理，那么可以这样定义handler：

DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;

* 1

**注意2：**  
标签的代码范围不包括declare handler的代码范围，所以在declare handler中不能使用iterate和leave语句，即使标签的范围包含了declare handler的范围。

在下述例子中，标签retry的范围是整个repeat循环的范围，在这个范围中使用了declare handler语句，表面上看retry包含了declare handler，但实际上retry的范围只是IF语句的范围，并不包括declare handler的范围。

CREATE PROCEDURE p ()

BEGIN

DECLARE i INT DEFAULT 3;

retry:

REPEAT

BEGIN

DECLARE CONTINUE HANDLER FOR SQLWARNING

BEGIN

ITERATE retry; # 我不属于retry的作用范围哦，所以我不能使用retry标签

END;

IF i < 0 THEN

LEAVE retry; #我才属于retry的范围，我可以使用retry标签。

END IF;

SET i = i - 1;

END;

UNTIL FALSE END REPEAT;

END;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

所以存储过程执行时会出现下述错误：  
ERROR 1308 (42000): LEAVE with no matching label: retry

所以为了避免在handler的中引用外部标签，可以使用下述方法：

1.定义exit类型的handler  
如果存储过程遇到异常停止运行时，无需做一些cleanup操作，可以如下定义：

DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;

* 1

如果需要做一些cleanup操作，可以在begin…end中编写相应处理逻辑:

DECLARE EXIT HANDLER FOR SQLWARNING

BEGIN

block cleanup statements

END;

* 1
* 2
* 3
* 4

2.定义continue类型的handler，并使用一个状态变量

CREATE PROCEDURE p ()

BEGIN

DECLARE i INT DEFAULT 3;

DECLARE done INT DEFAULT FALSE;

retry:

REPEAT

BEGIN

DECLARE CONTINUE HANDLER FOR SQLWARNING

BEGIN

SET done = TRUE;

END;

IF done OR i < 0 THEN

LEAVE retry;

END IF;

SET i = i - 1;

END;

UNTIL FALSE END REPEAT;

END;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18

这里使用了一个名称为done的变量，通过判断这个变量的状态，从而得知是否调用了continue handler。

**3.diagnostics area**

SQL语句的执行会产生诊断信息，并存放于诊断区中  
通过GET DIAGNOSTICS语句获取诊断区中的内容，该语句不需要特殊的权限。

诊断区分为当前诊断区和堆栈诊断区，通过CURRENT关键字来获取当前诊断区中的内容，通过STACKED获取堆栈诊断区中的内容，堆栈诊断区只有在上下文为condition handler的情况下才可以使用，如果不指定关键字默认从当前诊断区获取信息

在客户端获取诊断区中的数据

DROP TABLE test.no\_such\_table;

GET DIAGNOSTICS CONDITION 1

@p1 = RETURNED\_SQLSTATE, @p2 = MESSAGE\_TEXT;

SELECT @p1, @p2;

* 1
* 2
* 3
* 4

此时并不能使用GET STACKED DIAGNOSTICS堆栈诊断区中的内容，  
因为GET STACKED DIAGNOSTICS只能在condition handler中使用

诊断区汇总包含2种信息：  
1.语句信息，例如conditions的数量和影响的行数  
2. Condition信息，包括错误代码和错误消息，如果SQL语句抛出多个 conditions，那么在这部分诊断区中，会为每一个condition分配一个condition区，如果没有抛出condition那么就不会分配

如果语句产生了3个condition，那么诊断区包含的语句信息和condition信息类似：

Statement information:

row count

... other statement information items ...

Condition Handling

Condition area list:

Condition area 1:

error code for condition 1

error message for condition 1

... other condition information items ...

Condition area 2:

error code for condition 2:

error message for condition 2

... other condition information items ...

Condition area 3:

error code for condition 3

error message for condition 3

... other condition information items ...

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

GET DIAGNOSTICS语句可以获取语句信息或者condition信息，但是一条GET DIAGNOSTICS无法同时获取这2种信息。

获取诊断区中语句信息并保存到p1和p2变量中，本例中获取的是condition的数量和rows-affected数量

GET DIAGNOSTICS @p1 = NUMBER, @p2 = ROW\_COUNT;

* 1

通过指定condition的编号获取诊断区中相应condition信息到p3和p4变量中，本例中获取的是sqlstate值和错误消息。

GET DIAGNOSTICS CONDITION 1

@p3 = RETURNED\_SQLSTATE, @p4 = MESSAGE\_TEXT;

* 1
* 2

在SQL标准中，如果出现多个condition，那么第一个condition是关于前一个SQL语句返回的sqlstate值的，但是在MySQL中，无法保证这一点，为了得到主要的错误，不能使用下面的方法：

GET DIAGNOSTICS CONDITION 1 @errno = MYSQL\_ERRNO;

* 1

而是先取回condition的数量，然后使用该值来指定要查看的condition  
正确的方法：

GET DIAGNOSTICS @cno = NUMBER;

GET DIAGNOSTICS CONDITION @cno @errno = MYSQL\_ERRNO;

* 1
* 2

关于诊断区，官网的例子：

CREATE PROCEDURE do\_insert(value INT)

BEGIN

-- Declare variables to hold diagnostics area information

DECLARE code CHAR(5) DEFAULT '00000';

DECLARE msg TEXT;

DECLARE rows INT;

DECLARE result TEXT;

-- Declare exception handler for failed insert

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

BEGIN

GET DIAGNOSTICS CONDITION 1

code = RETURNED\_SQLSTATE, msg = MESSAGE\_TEXT;

END;

-- Perform the insert

INSERT INTO t1 (int\_col) VALUES(value);

-- Check whether the insert was successful

IF code = '00000' THEN

GET DIAGNOSTICS rows = ROW\_COUNT;

SET result = CONCAT('insert succeeded, row count = ',rows);

ELSE

SET result = CONCAT('insert failed, error = ',code,', message = ',msg);

END IF;

-- Say what happened

SELECT result;

END;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

假设上述存储过程中的t1表的字段类型int，并且not null,那么在进行下述操作分别向表t1中插入非空值和空值，各自得到的结果如下：

#插入非空值

mysql> CALL do\_insert(1);

+---------------------------------+

| result |

+---------------------------------+

| insert succeeded, row count = 1 |

+---------------------------------+

##插入null

mysql> CALL do\_insert(NULL);

+---- ------------------------------------------------------------+

| result |

+-----------------------------------------------------------------+

insert failed, error = 23000, message = Column 'int\_col' cannot be null

+-----------------------------------------------------------------+

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15

当存储过程中的condition handler被激活时，会发生一个向诊断区堆栈推送的事件：  
1.当前诊断区(第一诊断区)会变为堆栈诊断区(第二诊断区)，并且创建一个新的诊断区作为当前诊断区。  
2.在condition Handler中可以使用 GET [CURRENT] DIAGNOSTICS 和 GET STACKED DIAGNOSTICS来获取当前诊断区或堆栈诊断区中的内容。  
3.在开始的时候，当前诊断区和堆栈诊断区会返回相同的结果,所以有可能从当前诊断区获取到被激活的Handler的condition的相关信息，只要此时handler中没有其他SQL语句去改变当前诊断区中的内容。  
4.随着Handler中语句的执行，会根据一定的规则对当前诊断区的内容进行清空或者修改。

所以更可靠地获取被激活condition handler中信息的方法是从堆栈诊断区中获取相关信息，因为堆栈诊断区中的内容不会被condition handler中的语句所修改，除了RESIGNAL语句。

通过下面例子来说明，在condition中如何通过 GET STACKED DIAGNOSTICS语句来获取关于handler异常的信息，尽管此时当前诊断区已经被清空或修改。

DROP TABLE IF EXISTS t1;

CREATE TABLE t1 (c1 TEXT NOT NULL);

DROP PROCEDURE IF EXISTS p;

delimiter //

CREATE PROCEDURE p ()

BEGIN

-- Declare variables to hold diagnostics area information

DECLARE errcount INT;

DECLARE errno INT;

DECLARE msg TEXT;

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

-- Here the current DA is nonempty because no prior statements

-- executing within the handler have cleared it

GET CURRENT DIAGNOSTICS CONDITION 1

errno = MYSQL\_ERRNO, msg = MESSAGE\_TEXT;

SELECT 'current DA before mapped insert' AS op, errno, msg;

GET STACKED DIAGNOSTICS CONDITION 1

errno = MYSQL\_ERRNO, msg = MESSAGE\_TEXT;

SELECT 'stacked DA before mapped insert' AS op, errno, msg;

-- Map attempted NULL insert to empty string insert

INSERT INTO t1 (c1) VALUES('');

-- Here the current DA should be empty (if the INSERT succeeded),

-- so check whether there are conditions before attempting to

-- obtain condition information

GET CURRENT DIAGNOSTICS errcount = NUMBER;

IF errcount = 0

THEN

SELECT 'mapped insert succeeded, current DA is empty' AS op;

ELSE

GET CURRENT DIAGNOSTICS CONDITION 1

errno = MYSQL\_ERRNO, msg = MESSAGE\_TEXT;

SELECT 'current DA after mapped insert' AS op, errno, msg;

END IF ;

GET STACKED DIAGNOSTICS CONDITION 1

errno = MYSQL\_ERRNO, msg = MESSAGE\_TEXT;

SELECT 'stacked DA after mapped insert' AS op, errno, msg;

END;

INSERT INTO t1 (c1) VALUES('string 1');

INSERT INTO t1 (c1) VALUES(NULL);

END;

//

delimiter ;

CALL p();

SELECT \* FROM t1;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50

在上述存储过程中，定义了一个condition handler，在这个handler的开头处分别获取当前诊断区和堆栈诊断区中的内容，然后执行一条insert语句，之后再分别查询当前诊断区和堆栈诊断区的内容。在handler定义结束后，是这个存储过程的主体，也就是2条insert语句，其中一条insert语句为非空值字符串，另外一条insert插入的值为null，所以该存储过程执行顺序如下：

1.首先成功执行INSERT INTO t1 (c1) VALUES(‘string 1’);  
2.执行INSERT INTO t1 (c1) VALUES(NULL);因为t1表中禁止插入空值，所以会抛出异常。  
3.抛出的异常被condition handler捕获，condition handler被激活从而触发其中的处理逻辑,并打印condition handler中当前诊断区和堆栈诊断区的内容，二者内容相同。  
4.condition handler中的INSERT INTO t1 (c1) VALUES(’’);语句执行，该语句的执行会清空当前诊断区中的内容。

+---------------------------------+-------+----------------------------+

| op | errno | msg |

+---------------------------------+-------+----------------------------+

| stacked DA before mapped insert | 1048 | Column 'c1' cannot be null |

+---------------------------------+-------+----------------------------+

* 1
* 2
* 3
* 4
* 5

5.再次打印condition handler中当前诊断区和堆栈诊断区的内容，此时因为当前诊断区的内容被清空，所以打印’mapped insert succeeded, current DA is empty’，接着打印堆栈诊断区中内容，因为堆栈诊断区中的内容不会随着语句的执行而被清空掉，所以堆栈诊断区显示的内容依旧是：

+--------------------------------+-------+----------------------------+

| op | errno | msg |

+--------------------------------+-------+----------------------------+

| stacked DA after mapped insert | 1048 | Column 'c1' cannot be null |

+--------------------------------+-------+----------------------------+

* 1
* 2
* 3
* 4
* 5

需要注意的是  
1.GET DIAGNOSTICS语句也会清空当前诊断区中的内容，所以上述代码中把condition handler中的insert语句去掉，得到的结果也是一样的

2.如果将上述存储过程进行如下修改，也就是将3条declare变量的语句放到  
declare handler中，实际的结果将取决于MySQL的版本，如果是在MySQL-5.7.2之前的版本，下述修改后并不会影响诊断区中的内容，实际结果与上述结果相同，如果实在MySQL-5.7.2及之后的版本，declare变量语句会清空当前诊断区中的内容。

CREATE PROCEDURE p ()

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

-- Declare variables to hold diagnostics area information

DECLARE errcount INT;

DECLARE errno INT;

DECLARE msg TEXT;

GET CURRENT DIAGNOSTICS CONDITION 1

errno = MYSQL\_ERRNO, msg = MESSAGE\_TEXT;

SELECT 'current DA before mapped insert' AS op, errno, msg;

GET STACKED DIAGNOSTICS CONDITION 1

errno = MYSQL\_ERRNO, msg = MESSAGE\_TEXT;

SELECT 'stacked DA before mapped insert' AS op, errno, msg;

...

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

所以在需要获取诊断区中的内容时，一定要从堆栈诊断区中获取，而不是当前诊断区。