Module07-03
数据库开发：MySQL Procedure

数据库开发 – MySQL Procedure

- SQL 语句
- Oracle PL/SQL
- ➔ MySQL Procedure
- C++ OTL
- 数据库建模工具

数据库开发 – MySQL Procedure

- 关于 MySQL 存储过程
  - 自 MySQL5.0 版本以后， MySQL 开始支持存储过程、函数和触发器
  - MySQL 的存储过程语法上支持 SQL2003 标准

MySQL Procedure - 语法

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

语法 - 创建存储过程和函数

- 创建存储过程和函数的语法

```
-- 创建存储过程
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

-- 创建函数
CREATE FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body
```

语法 - 创建存储过程和函数

- 创建存储过程和函数的语法细节

```
--  存储过程的参数
proc_parameter:
    [ IN | OUT | INOUT ] param_name type

--  函数的参数（注意函数的参数默认为是 IN 的，没有 OUT 和 INOUT ）
func_parameter:
    param_name type

    type:    --  参数类型
        Any valid MySQL data type

characteristic:  --  存储过程或函数的特征
    LANGUAGE SQL
    | [NOT] DETERMINISTIC
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES
SQL DATA }
    | COMMENT 'string'

routine_body:  --  存储过程和函数体
    Valid SQL procedure statement or statements
```

语法 - 创建存储过程和函数

- 创建存储过程示例

```
delimiter //

CREATE PROCEDURE simpleproc (OUT param1 INT)
BEGIN
    SELECT COUNT(*) INTO param1 FROM Employee;
END
//

delimiter ;
```

  - 注意事项：
    - 上面语句中，前后 2 次执行 delimiter 指令，是为了避免过程体中的语句提前结束（如上面的 SELECT 语句是以引号 ';' ）

语法 - 创建存储过程和函数

- 创建函数示例

```
delimiter //

CREATE FUNCTION hello (s CHAR(20))
    RETURNS CHAR(50)
        RETURN CONCAT('Hello, ', s, '!');
//

delimiter ;
```

```
-- 同上
delimiter //

CREATE FUNCTION hello (s CHAR(20))
    RETURNS CHAR(50)
BEGIN
    RETURN CONCAT('Hello, ', s, '!');
END//

delimiter ;
```

语法 - 调用存储过程或函数

- CALL 语句
  - 调用已存在的存储过程和函数，语法如下：

```
CALL sp_name([parameter[,...]])
```

  - ALL 语句可以用声明为 OUT 或的 INOUT 参数的参数给它的调用者传回值。它也“返回”受影响的行数，客户端程序可以在 SQL 级别通过调用 ROW_COUNT() 函数获得这个数，从 C 中是调用 the mysql_affected_rows() C API 函数来获得。

语法 - 调用存储过程或函数

- CALL 语句示例

```
-- 调用存储过程 simpleproc
-- 将结果存放在变量 a 中
CALL simpleproc(@a);

-- 查看变量 a 的值
SELECT @a;
```

```
-- 调用函数 hello
-- 传入参数 'world'
-- 预期的输出: hello, world!
SELECT hello('world');
```

语法 - 修改、删除存储过程或函数

- 修改存储过程和函数的特征
  - ALTER 语句只用于修改存储过程和函数的特征 (characteristic)
  - 修改语法如下:

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic:
    { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
    | SQL SECURITY { DEFINER | INVOKER }
    | COMMENT 'string'
```

- 删除存储过程和函数

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

语法 - 查看存储过程或函数内容

- 查看存储过程和函数的内容
  - 语法：

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

  - 示例：

```
show create procedure simpleproc\G -- \G 表示纵向输出结果
*********************** 1. row ***********************
           Procedure: simpleproc
            sql_mode:
    Create Procedure: CREATE DEFINER=`root`@`localhost`
PROCEDURE `simpleproc`(OUT param1 INT)
BEGIN
    SELECT COUNT(*) INTO param1 FROM Employee;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
1 row in set (0.00 sec)
```

语法 - 查看存储过程或函数状态

- 查看存储过程和函数的状态
  - 语法：

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

  - 示例：

```
show procedure status like 'simple%'\G  -- \G 表示纵向输出结果
*********************** 1. row ***********************
                  Db: attendance_sys
                Name: simpleproc
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2010-05-13 16:03:05
             Created: 2010-05-13 16:03:05
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
```

MySQL Procedure - 复合语句块

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

复合语句块 - BEGIN..END

- 语法：

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

  - 说明：
    - 存储子程序可以使用 BEGIN ... END 复合语句来包含多个语句。 statement_list 代表一个或多个语句的列表。 statement_list 之内每个语句都必须用分号（；）来结尾。
    - 复合语句可以被标记。除非 begin_label 存在，否则 end_label 不能被给出，并且如果二者都存在，他们必须是同样的。

复合语句块 - BEGIN..END

- 复合语句示例

```
outer_blk: BEGIN

    inner_blk: BEGIN
        SELECT * FROM Department;
    END inner_blk;

    SELECT * FROM Employee;

END outer_blk;
```

MySQL Procedure - 变量声明

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

声明 - 关于声明

- 关于声明
  - DECLARE 语句被用来把不同项目归入到一个子程序：局部变量，条件和处理程序及游标。
  - DECLARE 仅被用在 BEGIN ... END 复合语句里，并且必须在复合语句的开头，在任何其它语句之前。
  - 游标必须在声明处理程序之前被声明，并且变量和条件必须在声明游标或处理程序之前被声明。

变量声明 - 示例

- 变量声明示例

```
CREATE PROCEDURE test_proc(IN arg INT)
BEGIN
    DECLARE v_salary    INT;
    DECLARE v_rate      FLOAT DEFAULT 0.75;
    DECLARE v_var       DOUBLE;
    DECLARE v_balance   NUMERIC(6,2);
    DECLARE reg_date    DATE DEFAULT '2000-01-01';
    DECLARE off_time    DATETIME DEFAULT '2000-01-01 16:00:00';
    DECLARE message1    CHAR(255) DEFAULT 'This will be padded
to 255 chars';
    DECLARE message2    VARCHAR(255) DEFAULT 'This will not be
padded';
    DECLARE content     TEXT DEFAULT 'This is a really
                        long string. In stored programs
                        we can use text columns fairly freely,
                        but in tables there are some
                        limitations regarding indexing
                        and use in various expressions.';
    /*  其它代码 */
END
```

变量声明 - 设置变量

- 为变量赋值： SET

```
DELIMITER $$

CREATE PROCEDURE test_proc2(OUT total INT)
BEGIN
    DECLARE v_salary    INT;
    SET v_salary = 3000;
    SET total = v_salary * 12;

    SELECT v_salary AS salary, total AS annual_sal;

    /*  其它代码 */
END$$

DELIMITER ;
```

MySQL Procedure - 游标

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

游标 - 隐式游标

- SELECT..INTO
  - 一个 SELECT..INTO 语句将产生一个隐式游标（或称简单游标），简单游标可以用于普通 sql 语句，也可以用于过程和函数

```
-- 普通 sql 语句
mysql> select count(*) into @n from Employee;
mysql> select @n; -- 查看结果
```

```
DELIMITER $$
CREATE PROCEDURE test_proc3()
BEGIN
    DECLARE v_salary    INT;
    SELECT max(salary) INTO v_salary
    FROM Employee;

    SELECT v_salary AS max_sal;
END$$

DELIMITER ;
```

游标 - 声明游标

- 游标声明语法

```
DECLARE cursor_name CURSOR FOR select_statement
```

  - 说明：
    - 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。
    - SELECT 语句不能有 INTO 子句

- 游标操作语法

```
--  打开游标: OPEN 语句

OPEN cursor_name


--  从游标中获取内容: FETCH 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。

FETCH cursor_name INTO var_name [, var_name] …


--  关闭游标: CLOSE 语句
--  如果未被明确地关闭，游标在它被声明的复合语句的末尾被关闭。

CLOSE cursor_name
```

游标 - 游标使用示例

- 示例

```
CREATE PROCEDURE test_cursor()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE dept_id VARCHAR(6);
  DECLARE max_sal INT;
  DECLARE cur1 CURSOR FOR
    SELECT depart_id, max(salary) FROM Employee
    GROUP BY depart_id;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

  OPEN cur1;
  REPEAT
    FETCH cur1 INTO dept_id, max_sal;
    IF NOT done THEN
       SELECT dept_id, max_sal;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
END;
```

MySQL Procedure - 流程控制

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

流程控制 - 概要

- MySQL 存储过程中流程控制语句
  - 分支语句
    - IF
    - CASE
  - 循环语句
    - LOOP
    - REPEAT
    - WHILE
    - LEAVE（跳出循环体，类似于 C++ 中的 break ）
    - ITERATE（继续循环，类似于 C++ 中的 continue ）

注：分支语句和循环语句都可以嵌套；
目前的版本尚不支持 FOR LOOP 。

流程控制 - IF 语句

- 语法

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

注意：关键字 **ELSEIF** 而不是 **ELSIF** 或 **ELSE IF**

流程控制 - IF 语句

- 示例

```
DELIMITER $$
DROP PROCEDURE IF EXISTS calc_level2$$
CREATE PROCEDURE calc_level2(IN eid varchar(8))
BEGIN
    DECLARE sal, level INT;
    SELECT salary INTO sal
    FROM Employee WHERE empl_id = eid;

    IF sal < 3000 THEN
        SET level = 1;
    ELSEIF sal > 3000 AND sal < 4000 THEN
        SET level = 2;
    ELSE
        SET level = 3;
    END IF;

    SELECT CONCAT('Level of ',eid,' is: ',level) result;
END$$
DELIMITER ;
```

流程控制 - CASE 语句

- 语法

```
-- Simple Case
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE

-- Searched Case
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

流程控制 - CASE 语句

- Simple Case 示例

```
DELIMITER $$
DROP PROCEDURE IF EXISTS guess_num$$

CREATE PROCEDURE guess_num(IN num INT)
BEGIN
    CASE num
        WHEN 12 THEN
            SELECT 'Good luck!';
        WHEN 64 THEN
            SELECT 'Good luck too!';
        ELSE
            SELECT 'Try again!';
    END CASE;
END$$
DELIMITER ;
```

流程控制 - CASE 语句

- Searched Case 示例

```
DELIMITER $$
DROP PROCEDURE IF EXISTS calc_level3$$
CREATE PROCEDURE calc_level3(IN eid varchar(8))
BEGIN
    DECLARE sal, level INT;
    SELECT salary INTO sal
    FROM Employee WHERE empl_id = eid;

    CASE
        WHEN sal < 3000 THEN
            SET level = 1;
        WHEN sal > 3000 AND sal < 4000 THEN
            SET level = 2;
        ELSE
            SET level = 3;
    END CASE;

    SELECT CONCAT('Level of ',eid,' is: ',level) result;
END$$
DELIMITER ;
```

流程控制 - LOOP 语句

- 语法

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

```
-- 跳出循环
LEAVE label

-- 继续循环
ITERATE label
```

  - 说明：
    - LOOP 语句一般借助 LEAVE 语句退出循环

流程控制 - LOOP 语句

- 示例

```
DELIMITER $$
DROP PROCEDURE IF EXISTS loop_test1$$
CREATE PROCEDURE loop_test1()
BEGIN
    DECLARE i INT;
    SET i = 0;
    lp: LOOP
        IF i > 20 THEN
            LEAVE lp;
        ELSEIF MOD(i, 3) = 0 THEN
            SET i = i + 2;
            ITERATE lp;
        ELSE
           SET i = i + 1;
        END IF;
    END LOOP lp;
END$$
DELIMITER ;
```

流程控制 - REPEAT 语句

- 语法

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

  - 说明：
    - REPEAT 语句也可以配合 LEAVE 和 ITERATE 语句使用

流程控制 - REPEAT 语句

- 示例

```
DELIMITER $$
DROP PROCEDURE IF EXISTS repeat_test1$$
CREATE PROCEDURE repeat_test1()
BEGIN
    DECLARE i INT;
    SET i = 0;
    REPEAT
        SET i = i + 1;
    UNTIL i > 10
    END REPEAT;
END$$
DELIMITER ;
```

流程控制 - WHILE 语句

- 语法

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

  - 说明：
    - WHILE 语句也可以配合 LEAVE 和 ITERATE 语句使用

流程控制 - WHILE 语句

- 示例

```
DELIMITER $$
DROP PROCEDURE IF EXISTS while_test1$$
CREATE PROCEDURE while_test1()
BEGIN
    DECLARE i  INT;
    SET i = 0;
    WHILE i <= 10 DO
        SET i = i + 1;
    END WHILE;
END$$

DELIMITER ;
```

MySQL Procedure - 错误处理

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

错误处理 - HANDLER

- 错误处理的语法

```
DECLARE handler_type HANDLER
    FOR condition_value[,...] sp_statement

handler_type:
    CONTINUE
    | EXIT
    | UNDO      --  目前尚不支持

condition_value:
    SQLSTATE [VALUE] sqlstate_value
    | condition_name
    | SQLWARNING
    | NOT FOUND
    | SQLEXCEPTION
    | mysql_error_code
```

错误处理 - HANDLER

- 语法说明
  - 错误处理器 (Handler) 类型：
    - EXIT ： 当前 BEGIN...END 复合语句的执行被终止，如果该语句块是最外层语句块，则退出子程序
    - CONTINUE ： 当前子程序的执行在执行处理程序语句之后继续
  - 条件值 (Condition value) ：
    - SQLWARNING 是对所有以 01 开头的 SQLSTATE 代码的速记。
    - NOT FOUND 是对所有以 02 开头的 SQLSTATE 代码的速记。
    - SQLEXCEPTION 是对所有没有被 SQLWARNING 或 NOT FOUND 捕获的 SQLSTATE 代码的速记。

错误处理 - 错误码

- 常用的错误码（续）

MySQL Error Code	SQLSTATE Code	错误信息
1011	HY000	Error on delete of '%s' (errno: %d)
1021	HY000	Disk full (%s); waiting for someone to free some space . . .
1022	23000	Can't write; duplicate key in table '%s'
1027	HY000	'%s' is locked against change
1036	HY000	Table '%s' is read only
1048	23000	Column '%s' cannot be null
1062	23000	Duplicate entry '%s' for key %d
1099	HY000	Table '%s' was locked with a READ lock and can't be updated
1100	HY000	Table '%s' was not locked with LOCK TABLES
1104	42000	The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay

错误处理 - 错误码

- 常用的错误码（续 1 ）

MySQL Error Code	SQLSTATE Code	错误信息
1106	42000	Incorrect parameters to procedure '%s'
1114	HY000	The table '%s' is full
1150	HY000	Delayed insert thread couldn't get requested lock for table %s
1165	HY000	INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES
1242	21000	Subquery returns more than 1 row
1263	22004	Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld
1264	22003	Out of range value adjusted for column '%s' at row %ld
1265	1000	Data truncated for column '%s' at row %ld
1312	0A000	SELECT in a stored program must have INTO
1317	70100	Query execution was interrupted
1319	42000	Undefined CONDITION: %s

- 常用的错误码（续2）

MySQL Error Code	SQLSTATE Code	错误信息
1319	42000	Undefined CONDITION: %s
1325	24000	Cursor is already open
1326	24000	Cursor is not open
1328	HY000	Incorrect number of FETCH variables
1329	2000	No data to FETCH
1336	42000	USE is not allowed in a stored program
1337	42000	Variable or condition declaration after cursor or handler declaration
1338	42000	Cursor declaration after handler declaration
1339	42000	Case not found for CASE statement
1348	HY000	Column '%s' is not updatable
1357	HY000	Can't drop a %s from within another stored routine
1358	HY000	GOTO is not allowed in a stored program handler
1362	HY000	Updating of %s row is not allowed in %s trigger
1363	HY000	There is no %s row in %s trigger

错误处理 - HANDLER 定义

- HANDLER 定义示例

```
-- 如在插入时发生主键重复错误，打印错误信息，然后继续
DECLARE CONTINUE HANDER FOR 1062
    SELECT 'Duplicate key in index';

-- 同上，使用相应的 SQLSTATE 代替 MySQL Error code
DECLARE CONTINUE HANDER FOR SQLSTATE '23000'
    SELECT 'Duplicate key in index';

-- 当游标中没数据可取出时，将 l_done=1，然后继续
DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET l_done=1;

-- 同上，使用相应的 SQLSTATE 代替 NOT FOUND
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
    SET l_done=1;

-- 同上，使用相应的 MySQL Error code 代替 SQLSTATE
DECLARE CONTINUE HANDLER FOR 1329
    SET l_done=1;
```

错误处理 - HANDLER 定义

- HANDLER 定义示例（续）

```
-- 如有任何错误发生，将 l_error 设为 1，然后继续
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SET l_error=1;

-- 如有任何错误发生，回滚事务，打印错误信息，退出语句块
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'Error occurred - terminating';
END;
```

错误处理 - 示例

- EXIT HANDLER 示例

```
CREATE PROCEDURE add_department (
        dept_id VARCHAR(6),
        dept_name VARCHAR(32),
        dept_location VARCHAR(45))
    MODIFIES SQL DATA  -- 存储过程的特征(characteristic)
BEGIN
    DECLARE duplicate_key INT DEFAULT 0;
    BEGIN -- 遇到1062错误则退出该语句块，而不是退出子程序
        -- Code 1062: Duplicate key
        DECLARE EXIT HANDLER FOR 1062 SET duplicate_key = 1;
        INSERT INTO Department (depart_id, name, location)
        VALUES (dept_id,dept_name,dept_location);
        SELECT CONCAT('Department ',dept_id,' created') Result;
    END;
    IF duplicate_key = 1 THEN
        SELECT CONCAT('Failed to insert ',dept_id,
                      ': duplicate key') Result;
    END IF;
END;
```

错误处理 - 示例

- CONTINUE HANDLER 示例

```
CREATE PROCEDURE add_department (
        dept_id VARCHAR(6),
        dept_name VARCHAR(32),
        dept_location VARCHAR(45))
    MODIFIES SQL DATA -- 存储过程的特征(characteristic)
BEGIN
    DECLARE duplicate_key INT DEFAULT 0;
    -- Code 1062: Duplicate key
    DECLARE CONTINUE HANDLER FOR 1062 SET duplicate_key = 1;
    INSERT INTO Department (depart_id, name, location)
    VALUES (dept_id,dept_name,dept_location);

    -- 如果遇到1062错误，则会转到此处，继续执行以下代码
    IF duplicate_key = 1 THEN
        SELECT CONCAT('Failed to insert ',dept_id,
                      ': duplicate key') Result;
    ELSE
        SELECT CONCAT('Department ',dept_id,' created') Result;
    END IF;
END;
```

错误处理 - 资源管理

- 资源管理
  - 各种资源在子程序退出之前必须能被正确释放，比如游标，使用之后必须关闭，而一旦遇到错误发生，游标的关闭操作往往无法保证，所以在错误处理的时候需考虑这些问题。
  - 下面是一个可能的方式：

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
    /*  一些错误处理代码 */
    CLOSE cur1; -- cur1 是之前打开的游标
END;
```

MySQL Procedure - 管理事务

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

管理事务 - 开始和结束事务

- 开始和结束事务的方式：
  - 事务的开始和结束由业务逻辑规定，如果几个操作之间有依赖关系，如后面的操作依赖于前面的操作的结果、或几个操作协同完成同一个事件，则需将这些操作放在一个事务中管理
  - 事务的开始，必须明确将数据库的自动提交 (autocommit) 机制关闭，在存储过程或函数中可以通过以下 2 种方式之一：
    - `SET AUTOCOMMIT = 0;` -- 或下面的语句
    - `START TRANSACTION;`
  - 如遇错误须用 ROLLBACK 回滚之前的操作，结束事务
  - 只有所有关联的操作均成功执行，才使用 COMMIT 提交所有操作，结束事务

管理事务 - 示例

- 开始和结束事务示例：

```
CREATE PROCEDURE transfer (
        account_from INT, account_to INT, amount NUMERIC(6,2))
BEGIN
    DECLARE has_error INT DEFAULT 0;
    DECLARE CONTINUE HANDLER
        FOR SQLEXCEPTION SET has_error = 1;
    SET AUTOCOMMIT = 0; -- 禁止自动提交事务，同: START TRANSACTION;
    UPDATE ACCOUNT
       SET balance = balance - amount
    WHERE account_id = account_from;
    UPDATE ACCOUNT
       SET balance = balance + amount
    WHERE account_id = account_to;
    IF has_error = 1 THEN
        ROLLBACK;  -- 如果有任何错误，须回滚事务
    ELSE
        COMMIT; -- 只有所有操作都成功执行，才提交事务
    END IF;
END;
```

MySQL Procedure - 触发器

- MySQL Procedure：
  - 语法
  - 复合语句块
  - 变量声明
  - 游标（Cursor）
  - 流程控制
  - 错误处理
  - 管理事务（Transaction）
  - 触发器（Trigger）

触发器 - 概要

- 关于触发器
  - 触发器 (Trigger) 是由某些数据库事件触发而执行的例程，我们可以将触发器将特定的数据库事件关联起来，当这些数据库事件发生时，触发器例程将被执行
  - MySQL 的触发器目前只支持 DML 事件：
    - 如对表执行 DELETE 、 INSERT 、 UPDATE 操作
  - 触发器的典型应用：
    - 操作的监控
    - 数据同步
    - ...

触发器 - 创建触发器

- CREATE TRIGGER 语法

```
CREATE TRIGGER trigger_name
    trigger_time trigger_event
    ON table_name FOR EACH ROW trigger_statement
```

触发器 - 创建触发器

- CREATE TRIGGER 说明
  - 触发程序与命名为 tbl_name 的表相关。 table_name 必须引用永久性表。不能将触发程序与临时表或视图关联起来。
  - trigger_time： 是触发程序的动作时间。它可以是 BEFORE 或 AFTER ，以指明触发程序是在激活它的语句之前或之后触发。
  - trigger_event： 指示激活触发程序的语句的类型。 trigger_event 可以是下述值之一：
    - INSERT： 将新行插入表时激活触发程序，例如，通过 INSERT 、 LOAD DATA 和 REPLACE 语句。
    - UPDATE： 更改某一行时激活触发程序，例如，通过 UPDATE 语句。
    - DELETE： 从表中删除某一行时激活触发程序，例如，通过 DELETE 和 REPLACE 语句。

触发器 - 示例

- 说明
  - 如同存储过程和函数，触发器能完成很多复杂的任务
  - 以下将通过一个示例来演示触发器的使用
  - 该示例将使用触发器来监控对表 Employee 的修改性操作
  - 针对 Employee 任意一次修改性的操作将被记录到表 OP_LOG

触发器 - 示例

- 示例相关的表

```
--  操作日志表
DROP TABLE IF EXISTS op_log;
CREATE TABLE op_log (
    log_seq     INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    who         VARCHAR(32)    NOT NULL,
    operation   VARCHAR(16)    NOT NULL,
    rec_id      VARCHAR(32)    NOT NULL,
    op_time     DATE           NOT NULL
);
```

触发器 - 示例

- 触发器定义示例：

```
CREATE TRIGGER trigger_empl_insert
    AFTER INSERT ON Employee
    FOR EACH ROW
BEGIN
    INSERT INTO op_log
    VALUES (NULL, USER(), 'insert', NEW.empl_id, SYSDATE());
END;
```

```
CREATE TRIGGER trigger_empl_update
    AFTER UPDATE ON Employee
    FOR EACH ROW
BEGIN
    INSERT INTO op_log
    VALUES (NULL, USER(), 'update', OLD.empl_id, SYSDATE());
END;
```

触发器 - 删除触发器

- DROP TRIGGER 语法

```
DROP TRIGGER trigger_name
```

MySQL Procedure - 更多资源

- MySQL Procedure 更多有用的资源：
  - MySQL 的官方文档： MySQL 5.1 Reference Manual
    - http://dev.mysql.com/doc/refman/5.1/en/index.html
  - 书籍： MySQL Stored Procedure Programming
    - 作者： Steven Feuerstein, Guy Harrison