MySQL学习笔记（Day014：触发器下/存储过程/自定义函数）

MySQL学习

一. 作业讲解

- **查询employees表中非基层用户的最近详细信息**

> **关于 `Group By` 在《SQL必知必会》中提及的部分规定：**
>
> 1. `GROUP BY` 子句中列出的每一列都必须是检索列或有效的表达式（但不能是聚集函数）。如果在SELECT中使用表达式，则必须在 `GROUP BY` 子句中指定相同的表达式。不能使用别名。
> 2. 除聚集计算语句外， `SELECT`语句中的每一列都必须在`GROUP BY`子句中给出 。

```
SELECT
    e.emp_no,
    CONCAT(last_name, ' ', first_name) AS name,
    t.title,
    dp.dept_name,
    s.salary
FROM
    employees e
        LEFT JOIN
    dept_manager d ON e.emp_no = d.emp_no
        LEFT JOIN
    (SELECT
        emp_no, title, from_date, to_date
    FROM
        titles
    WHERE
        (emp_no , from_date, to_date) IN (SELECT
                emp_no, MAX(from_date), MAX(to_date)
            FROM
                titles AS b
            GROUP BY b.emp_no)) t ON t.emp_no = e.emp_no
        LEFT JOIN
    (SELECT
        dept_no, emp_no, from_date, to_date
    FROM
        dept_emp
    WHERE
        (emp_no , from_date, to_date) IN (SELECT
                emp_no, MAX(from_date), MAX(to_date)
            FROM
                dept_emp AS b
            GROUP BY b.emp_no)) de ON de.emp_no = e.emp_no
        LEFT JOIN
    (SELECT
        emp_no, salary, from_date, to_date
    FROM
        salaries
    WHERE
        (emp_no , from_date, to_date) IN (SELECT
                emp_no, MAX(from_date), MAX(to_date)
            FROM
                salaries AS b
            GROUP BY b.emp_no)) s ON s.emp_no = e.emp_no
        LEFT JOIN
    departments dp ON dp.dept_no = de.dept_no
WHERE
    d.emp_no IS NULL;


--
-- 改进的子查询语句 - 1
--
SELECT
    emp_no, title, from_date, to_date
FROM
    titles
        WHERE
        (emp_no , from_date, to_date) IN
            (
                SELECT
                    emp_no, MAX(from_date), MAX(to_date)  -- 因为数据本身的问题，这里from_date和to_date都要
                FROM
                    titles AS b
                    GROUP BY b.emp_no
            ) -- 这个子查询表示以emp_no分类，找到最大（最近）的from_date和to_date
              -- 而where条件在这个最大的基础上，过滤出我们要的title。(salary同理)


--
-- 改进的子查询语句 - 2
--
SELECT
    emp_no, title, from_date, to_date
FROM
    titles AS a
WHERE
    (from_date, to_date) = (SELECT
            MAX(from_date), MAX(to_date)  -- 同样使用from_date和to_date
        FROM
            titles AS b
        WHERE
            a.emp_no = b.emp_no  -- 这个是一个关联子查询
        GROUP BY b.emp_no);
```

- **Rank排名一条SQL语句**

```
mysql> select * from test_rank_2;
+------+-------+
| id   | score |
+------+-------+
|    1 |    10 |
|    2 |    20 |
|    3 |    30 |
|    4 |    30 |
|    5 |    40 |
|    6 |    40 |
+------+-------+
6 rows in set (0.00 sec)

mysql> select id, score,
    -> case
    ->     when @prev_value = score then @rank_count
    ->     when @prev_value := score then @rank_count := @rank_count + 1
    -> end as rank_column
    -> from test_rank_2, (select @prev_value:=NULL, @rank_count:=0) as t  -- 和RowNumber思路一样，增加一个表
    -> order by score desc;
+------+-------+-------------+
| id   | score | rank_column |
+------+-------+-------------+
|    5 |    40 | 1           |
|    6 |    40 | 1           |
|    3 |    30 | 2           |
|    4 |    30 | 2           |
|    2 |    20 | 3           |
|    1 |    10 | 4           |
+------+-------+-------------+
6 rows in set (0.00 sec)
```

二. 触发器 · 下

1. 触发器总结

- 触发器对性能有损耗，应当非常慎重使用；
- 对于事物表， 触发器执行失败则整个语句回滚 ；
- Row格式主从复制， 触发器不会在从库上执行 ；
  - 因为从库复制的肯定是主库已经提交的数据，既然已经提交了说明触发器已经被触发过了，所以从库不会执行。
- 使用触发器时应防止递归执行；

```
delimiter //
create trigger trg_test
    before update on 'test_trigger'
    for each row
begin
    update test_trigger set score=20 where name = old.name;  -- 又触发了update操作，循环触发了
end;//
```

2. 触发器模拟物化视图

- **物化视图的概念**
  - 不是基于基表的虚表
  - 根据基表实际存在的实表
  - 预先计算并保存耗时较多的SQL操作结果（如多表链接(join)或者group by等）
- **模拟物化视图**

```
mysql> create table Orders
    -> (order_id int unsigned not null auto_increment,
    -> product_name varchar(30) not null,
    -> price decimal(8,2) not null,
    -> amount smallint not null,
    -> primary key(order_id));
Query OK, 0 rows affected (0.13 sec)  -- 创建Orders表

mysql> insert into Orders values
    -> (null, 'cpu', 135.5 ,1),
    -> (null, 'memory', 48.2, 3),
    -> (null, 'cpu', 125.6, 3),
    -> (null, 'cpu', 105.3, 4);
Query OK, 4 rows affected (0.06 sec)  -- 插入测试数据
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from  Orders;
+----------+-------------+--------+--------+
| order_id | product_name | price  | amount |
+----------+-------------+--------+--------+
|        1 | cpu          | 135.50 |      1 |
|        2 | memory       |  48.20 |      3 |
|        3 | cpu          | 125.60 |      3 |
|        4 | cpu          | 105.30 |      4 |
+----------+-------------+--------+--------+
4 rows in set (0.00 sec)

-- 建立一个模拟物化视图的表（即用这张表来模拟物化视图）
mysql> create table Orders_MV
    -> ( product_name varchar(30) not null,
    ->  price_sum decimal(8,2) not null,
    ->  amount_sum int not null,
    ->  price_avg float not null,
    ->  orders_cnt int not null,
    ->  unique index (product_name));
Query OK, 0 rows affected (0.14 sec)

-- 通过Orders表的数据，将测试数据初始化到Orders_MV表中
mysql> insert into Orders_MV
    ->     select product_name, sum(price),
    ->            sum(amount), avg(price), count(*)
    ->     from Orders
    ->     group by product_name;
Query OK, 2 rows affected (0.07 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from Orders_MV;
+-------------+-----------+------------+-----------+------------+
| product_name | price_sum | amount_sum | price_avg | orders_cnt |
+-------------+-----------+------------+-----------+------------+
| cpu          |    366.40 |          8 |   122.133 |          3 |
| memory       |     48.20 |          3 |      48.2 |          1 |
+-------------+-----------+------------+-----------+------------+
2 rows in set (0.00 sec)

-- 在MySQL workbench中输入，比较方便
delimiter //

CREATE TRIGGER tgr_Orders_insert -- 创建触发器为tgr_Orders_insert
    AFTER INSERT ON Orders  -- 触发器是INSERT类型的，且作用于Orders表
    FOR EACH ROW
BEGIN
    SET @old_price_sum := 0;  -- 设置临时存放Orders_MV表(模拟物化视图)的字段的变量
    SET @old_amount_sum := 0;
    SET @old_price_avg := 0;
    SET @old_orders_cnt := 0;
    SELECT   -- select ... into ... 在更新Orders_MV之前，将Orders_MV中对应某个产品的信息写入临时变量
        IFNULL(price_sum, 0),
        IFNULL(amount_sum, 0),
        IFNULL(price_avg, 0),
        IFNULL(orders_cnt, 0)
    FROM
        Orders_MV
    WHERE
        product_name = NEW.product_name INTO @old_price_sum , @old_amount_sum , @old_price_avg , @old_orders_cnt;

    SET @new_price_sum = @old_price_sum + NEW.price; -- 累加新的值
    SET @new_amount_sum = @old_amount_sum + NEW.amount;
    SET @new_orders_cnt = @old_orders_cnt + 1;
    SET @new_price_avg = @new_price_sum / @new_orders_cnt ;

    REPLACE INTO Orders_MV
            VALUES(NEW.product_name, @new_price_sum,
                   @new_amount_sum, @new_price_avg, @new_orders_cnt );
   -- REPLACE 将对应的物品（唯一索引）的字段值替换new_xxx的值
END;//

delimiter ;


mysql> insert into Orders values (null, 'ssd', 299, 3);
Query OK, 1 row affected (0.03 sec)

mysql> insert into Orders values (null, 'memory', 47.9, 5);
Query OK, 1 row affected (0.05 sec)

mysql> select * from Orders_MV;
+-------------+-----------+------------+-----------+------------+
| product_name | price_sum | amount_sum | price_avg | orders_cnt |
+-------------+-----------+------------+-----------+------------+
| cpu          |    366.40 |          8 |   122.133 |          3 |
| memory       |     96.10 |          8 |     48.05 |          2 | -- 数量自动增加了1，价格也发生了变化
| ssd          |    299.00 |          3 |       299 |          1 | -- 新增加的ssd产品
+-------------+-----------+------------+-----------+------------+
3 rows in set (0.00 sec)


--
-- IFNULL MySQL内建函数的演示
--
mysql> select @test;
+-------+
| @test |
+-------+
| NULL  |  -- 当前会话中没有test变量
+-------+
1 row in set (0.00 sec)

mysql> select ifnull(@test, 100);   -- 如果test为NULL，则ifnull返回100
+--------------------+
| ifnull(@test, 100) |
+--------------------+
| 100                |  -- ifnull函数return的值是100
+--------------------+
1 row in set (0.00 sec)

mysql> select @test;
+-------+
| @test |
+-------+
| NULL  |  -- 但是test还是NULL
+-------+
1 row in set (0.00 sec)

mysql> set @test:=200;  -- 给test变量赋值为200
Query OK, 0 rows affected (0.00 sec)

mysql> select ifnull(@test, 100);  -- 再次ifnull判断，此时test不为null，则返回test变量的值
+--------------------+
| ifnull(@test, 100) |
+--------------------+
|                200 |  -- test不为null。返回test的值200
+--------------------+
1 row in set (0.00 sec)

--
-- select into 用法
--
mysql> select @id_1;
+-------+
| @id_1 |
+-------+
|  NULL | -- 当前变量id_1为null
+-------+
1 row in set (0.00 sec)

mysql> select @score_1;
+----------+
| @score_1 |
+----------+
|     NULL |  -- 当前变量score_1为null
+----------+
1 row in set (0.00 sec)

mysql> select * from test_rank_2;
+------+-------+
| id   | score |
+------+-------+
|    1 |    10 |
|    2 |    20 |
|    3 |    30 |
|    4 |    30 |
|    5 |    40 |
|    6 |    40 |
+------+-------+
6 rows in set (0.00 sec)

mysql> select * from test_rank_2
    -> where id=1 into @id_1, @score_1;
-- 选择id=1的记录，将对应的id和score赋值给变量 id_1 和 score_1
Query OK, 1 row affected (0.00 sec)

mysql> select @id_1;
+-------+
| @id_1 |
+-------+
|     1 |
+-------+
1 row in set (0.00 sec)

mysql> select @score_1;
+----------+
| @score_1 |
+----------+
|       10 |
+----------+
1 row in set (0.00 sec)

-- 触发器对性能会有影响，相当于在一个事物中插入了其他的事物
```

三. 存储过程

1. 存储过程介绍

- 存储在数据库端的一组SQL语句集；
- 用户可以通过存储过程名和传参多次调用的程序模块；
- 存储过程的特点：
  - 使用灵活，可以使用流控语句、自定义变量等完成复杂的业务逻辑；
  - 提高数据安全性，屏蔽应用程序直接对表的操作，易于进行审计；
  - 减少网络传输；
  - 提高代码维护的复杂度，实际使用需要结合业务评估；

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body


proc_parameter:  -- 注意，只有procedure才有in(传入),out(传出),inout(传入传出)参数，自定义函数（只有）默认就是 in。
    [ IN | OUT | INOUT ] param_name type


characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement


-- 删除
DROP PROCEDURE  procedure_name;
```

2. 存储过程举例与流程控制语句

> 流程控制语句 官方文档

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

-- 例子
mysql> delimiter //
mysql> create procedure pcd_test_1 (in param_a int) -- 创建一个
    -> begin
    -> declare a int;  -- delcare声明了该变量的作用域在该procedure中
    ->     if param_a > 10 then set a:=11;
    ->         elseif param_a = 10 then set a:=10;
    ->         else set a:=9;
    ->     end if;
    -> end;//
Query OK, 0 rows affected (0.01 sec)

mysql> select @a;  -- 查看当前会话中变量a的值
+------+
| @a   |
+------+
| NULL |  -- 当前会话中a为NULL
+------+
1 row in set (0.00 sec)

mysql> call pcd_test_1(1);
+------+
| a    |
+------+
|    9 |
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call pcd_test_1(10);
+------+
| a    |
+------+
|   10 |
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call pcd_test_1(20);
+------+
| a    |
+------+
|   11 |
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> select @a;
+------+
| @a   |
+------+
| NULL | -- 使用了declare，使得procedure中a的作用域限制在了procedure内
+------+
1 row in set (0.00 sec)

--
-- CASE WHEN
--
-- CASE WHEN 语法
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
-- 或者是
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE

--
-- CASE WHEN 例子
--
mysql> delimiter //
mysql>
mysql> create procedure pcd_test_2(in param_1 int)
    ->   begin
    ->     case param_1
    -- 当case后面有value时，该value会和when中的when_value进行"="判断
    -- 相等则执行then后面的语句，然后跳出；否则就进行下一次when的匹配
    ->       when 2 then select 200;
    ->       when 3 then select 300;
    ->       else
    ->         begin
                    -- 当没有匹配时，且else中没有要执行的语句
                    -- 则给一个begin/end的空语句；
                    -- 或者不写else语句；
    ->         end;
    ->     end case;
    -> end;//
Query OK, 0 rows affected (0.03 sec)

mysql> delimiter ;
mysql> call pcd_test_2(1);
Query OK, 0 rows affected (0.00 sec)

mysql> call pcd_test_2(2);
+-----+
| 200 |
+-----+
| 200 |
+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call pcd_test_2(3);
+-----+
| 300 |
+-----+
| 300 |
+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

-- 另外一种SQL语法请参考rank排名作业：注意when后跟的是condition

--
-- WHILE  循环
--
-- WHILE 语法
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]

-- WHILE举例
mysql> delimiter //
mysql>
mysql> create procedure pcd_test_3(in param_1 int)
    -> begin
    ->     declare a int default 1;
    ->     while param_1 > 10 do
    ->         set param_1 = param_1 - 1;
    ->         set a = a + 1;
    ->     end while;
    ->     select a;
    -> end;//
Query OK, 0 rows affected (0.01 sec)
mysql> delimiter ;

mysql> call pcd_test_3(15); -- 15 - 10 = 5：需要5次循环
+------+
| a    |
+------+
|    6 |  -- a + 5 = 6
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)


--
-- REPEAT   循环
--
-- REPEAT 语法
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]

mysql> delimiter //
mysql> create procedure pcd_test_4(in param_1 int)
    -> begin
    ->     SET @x = 0;  -- 没有使用declare，所以x是会话级别的
    ->     REPEAT
    ->         SET @x = @x + 1;
    ->     UNTIL @x > param_1 END REPEAT;
    -> end;//
Query OK, 0 rows affected (0.01 sec)
mysql> delimiter ;

mysql> call pcd_test_4(10);
Query OK, 0 rows affected (0.00 sec)

mysql> select @x; -- x是会话级别的
+------+
| @x   |
+------+
|   11 |  -- 一共循环11次(10>10 为False，11 > 10为True，才跳出)
+------+
1 row in set (0.00 sec)

--
-- loop 循环
```

```
--
-- loop语法
[begin_label:] LOOP
    statement_list
END LOOP [end_label]

-- ITERATE 和label相结合，表示继续从label处执行
-- LEAVE   和label相结合，表示从label 标记的代码段离开

-- loop 例子
mysql> delimiter //
mysql> create procedure pcd_test_5(in param_1 int)
    -> begin
    ->     test_label: loop
    ->         set param_1 := param_1 + 1; -- 参数累加
    ->         if param_1 < 10 then        -- 如果累加的值小于10
    ->             iterate test_label;     -- 继续执行 标签 test_label
    ->         end if;
    ->         leave test_label;  -- 如果>=10则离开这个test_label(loop)
    ->     end loop test_label;
    ->     set @x = param_1; -- 设置会话级别的变量
    -> end;//

Query OK, 0 rows affected (0.02 sec)
mysql> delimiter ;

mysql> call pcd_test_5(5); -- 5<10 ，累加5次后>=10为true，离开循环
Query OK, 0 rows affected (0.00 sec)

mysql> select @x;
+------+
| @x   |
+------+
|   10 |  -- 累加到10的 param_1 赋值给 x，即为10
+------+
1 row in set (0.00 sec)


-- 老师给出的例子， 阶乘
mysql> create table test_proc_1(a int, b int); -- 给一个存放数据的表
Query OK, 0 rows affected (0.15 sec)

mysql> delimiter //
mysql> create procedure proc_test1(in total int, out res int)
    -> begin
    ->     declare i int;
    ->     set i := 1;
    ->     set res := 1;
    ->     if total <= 0 then
    ->         set total := 1;
    ->     end if;
    ->     while i <= total do
    ->         set res := res * i;
    ->         insert into test_proc_1 values(i, res);
    ->         set i := i + 1;
    ->     end while;
    -> end;//
Query OK, 0 rows affected (0.01 sec)
mysql> delimiter ;

mysql> set @res_value := 0;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc_test1(5, @res_value); -- 因为res是out变量，要预先有这个变量，这里上面设置了res_value(实参和形参不必同名)
Query OK, 1 row affected (0.15 sec)

mysql> select @res_value;
+------------+
| @res_value |
+------------+
|        120 | -- 5的阶乘的结果是120
+------------+
1 row in set (0.00 sec)

mysql> select * from test_proc_1;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    6 |
|    4 |   24 |
|    5 |  120 |  -- 每次insert的结果
+------+------+
5 rows in set (0.00 sec)
```

三. 自定义函数

- 自定义函数和存储过程很类似，但是必须要有返回值；
- 与内置的函数(sum(), max()等)使用方法类似
  - select fun(val);
  - select * from t where col= fun(val);
- 自定义函数可能在遍历每条记录中使用；

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type  -- 必须有返回值
    [characteristic ...] routine_body

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement

-- 删除
DROP FUNCTION fun_name;
```

```
-- 老师给的例子，还是阶乘，用自定义函数的方式

mysql> delimiter //
mysql>
mysql> create function fun_test_1(total int)
    -> returns int
    -> begin
    ->     declare i int;
    ->     declare res int;
    ->     set i := 1;
    ->     set res := 1;
    ->     if total <= 0 then
    ->         set total := 1;
    -> end if;
    ->     while i <= total do
    ->         set res := res * i;
    ->         set i := i + 1;
    ->     end while;
    ->     return res;
    -> end;//
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
-- 报错，提示因为函的声明中没有"DETERMINISTIC, NO SQL, or READS SQL DATA"等关键字 ，需要使用打开参数 log_bin_trust_function_creators

-- 解决方法，set global log_bin_trust_function_creators=1; 开启该选项可能会引起主从服务器不一致
--           或者 增加 上述相应功能的关键字


-- 使用 deterministic 关键字
-- 当你声明一个函数的返回是确定性的，则必须显示的使用deterministic关键字，默认是 no deterministic的
mysql> delimiter //
mysql> create function fun_test_1(total int)
    -> returns int deterministic -- 这个只是告诉MySQL我这个函数是否会改变数据
                                 -- 即使我下面使用了insert，update等DML语句，MySQL不会检查
                                 -- 函数是否会改变数据，完全依赖创建函数的用户去指定的关键字
                                 -- 而非真的是否有修改数据
                                 -- 只是声明，而非约束
    -> begin
    ->     declare i int;
    ->     declare res int;
    ->     set i := 1;
    ->     set res := 1;
    ->     if total <= 0 then
    ->         set total := 1;
    ->     end if;
    ->     while i <= total do
    ->         set res := res * i;
    ->         insert into test_proc_1 values(i, res);  -- 在自定义函数中，同样可以使用sql
                                                        -- 并且该SQL是insert，其实和deterministic违背。
    ->         set i := i + 1;
    ->     end while;
    ->     return res;
    -> end;//
Query OK, 0 rows affected (0.01 sec)
mysql> delimiter ;

mysql> truncate table test_proc_1;
Query OK, 0 rows affected (0.11 sec)

mysql> select fun_test_1(6);  -- return了6的阶乘，720
+---------------+
| fun_test_1(6) |
+---------------+
|           720 |
+---------------+
1 row in set (0.02 sec)

mysql> select * from test_proc_1;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    6 |
|    4 |   24 |
|    5 |  120 |
|    6 |  720 |  -- 使用了insert语句进行插入阶乘的历史记录
+------+------+
6 rows in set (0.00 sec)

-- 关键字简单说明
-- DETERMINISTIC ：  当给定相同的输入，产生确定的结果
-- NOT DETERMINISTIC ：  默认值，认为产生的结果是不确定的

-- READS SQL DATA  ：  只是读取SQL数据
-- MODIFIES SQL DATA ：  会修改数据
-- NO SQL ：  没有SQL遇见
-- CONTAINS SQL ：  包含SQL语句，但是没有读写语句，理论有select now()等
```

原文链接

部分原文：

By default, for a CREATE FUNCTION statement to be accepted, at least one of DETERMINISTIC, NO SQL, or READS SQL DATA must be specified explicitly. Otherwise an error occurs:

默认情况下，在创建自定义函数时，必须显示的指定关键字 `DETERMINISTIC, NO SQL, 或者是 READS SQL DATA` 中的至少一个(可以多个)，否则就会有如下错误

> ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

网上参考资料1

网上参考资料2